2005

Biologically inspired learning system

Patrick McDowell
Louisiana State University and Agricultural and Mechanical College, patrick.mcdowell@nrlssc.navy.mil

BIOLOGICALLY INSPIRED LEARNING SYSTEM

A Dissertation
Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

By
Patrick McDowell
B.S., University of Idaho, 1984
M.S., University of Southern Mississippi, 2000
December, 2005

Acknowledgements

I would like to thank Dr. S. S. Iyengar for giving me the opportunity to pursue my interests at LSU. From working on the robo-tiger to completing this dissertation, his support, advice, and friendship over the years have been invaluable.

I would also like to thank my committee members, Dr. Jianhua Chen, Dr. John Tyler, Dr. Li Li, Dr. Jim Belanger, and Dr. Brian Bourgeois. They provided advice, encouragement and technical guidance throughout the dissertation process.

My colleagues at the Naval Research Laboratory provided unwavering support through the years of this work and during the time that I was hospitalized and recovering. I would like to recognize Dr. Herb Eppert Jr., Mike Harris, and Dr. Stanley Chin-Bing for supporting my wife and I during this endeavor. My boss, Dr. Brian Bourgeois was a consistent source of help, support, and friendship, as were my coworkers, Marvin Roe, Frank McCreedy, Don Brandon, Marlin Gendron, and John Sample.

I want to thank my mom and pop, my sister Stephanie, and my friend Mark Bland, they were just a phone call away when I was tired, needed help or encouragement. And not to be left out, my fuzzy feline friends that provided so much inspiration for this work, Heidi, LaLa, Rocky, Mizty, Tiger, Zee, Twirly, Max, and the rest of the gang.

And lastly, special thanks goes to my sweet, beautiful, wonderful and nutty wife Pamela Marie for being with me throughout this adventure.

Table of Contents

Abstract

Learning Systems used on robots require either a-priori knowledge in the form of models, rules of thumb or databases or require that robot to physically execute multitudes of trial solutions. The first requirement limits the robot's ability to operate in unstructured changing environments, and the second limits the robot's service life and resources. In this research a generalized approach to learning was developed through a series of algorithms that can be used for construction of behaviors that are able to cope with unstructured environments through adaptation of both internal parameters and system structure as a result of a goal based supervisory mechanism. Four main learning algorithms have been developed, along with a goal directed random exploration routine. These algorithms all use the concept of learning from a recent memory in order to save the robot/agent from having to exhaustively execute all trial solutions. The first algorithm is a reactive online learning algorithm that uses a supervised learning to find the sensor/action combinations that promote realization of a preprogrammed goal. It produces a feed forward neural network controller that is used to control the robot. The second algorithm is similar to first in that it uses a supervised learning strategy, but it produces a neural network that considers past values, thus providing a non-reactive solution. The third algorithm is a departure from the first two in that uses a non-supervised learning technique to learn the best actions for each situation the robot encounters. The last algorithm builds a graph of the situations encountered by agent/robot in order to learn to associate the best actions with sensor inputs. It uses an unsupervised learning approach based on shortest paths to a goal situation in the graph in order to generate a non-reactive feed forward neural network. Test results were good, the first and third algorithms were tested in a formation maneuvering task in both simulation and onboard mobile robots, while the second and fourth were tested simulation.

Chapter 1. Introduction

Robots intended for autonomous operation in unstructured environments, i.e. changing environments with unknown attributes, provide several challenges for their designers. For example, a robot needs to be able to consistently make correct decisions from information acquired through sensors whose performance will change over time. Also, given that the operational environment cannot be completely known to it's designers, the robot will need a mechanism to modify or add to its repertoire of behaviors as it encounters new situations. It will also need to be able to change its behaviors in order to achieve its designed goals while it continues its moment-to-moment operation. Natural selection and reproduction have allowed many species of animals to endure the rigors of changing environments[1], but limits in current technologies do not afford this capability to robots. This requires that the robot be able adapt to the environment without overtaxing the limited service life of its equipment so that it can behave in a manner that satisfies the preprogrammed goals.

Most animals have the ability to learn[2] new behaviors in order to fulfill their immediate goals. In 1898 Edward Thorndike did a series of experiments in which a hungry cat was placed in a cage consisting of a slatted box and shown a bowl of food. For the cat to eat the food, it had to figure out how to engage a mechanism that opened the cage. The cat would initially display a variety of behaviors, including trying to reach through the slats to the food, scratching at parts of the cage and so on. Eventually the cat would accidentally hit the release mechanism and escape the cage and eat the food. On each subsequent session the cat would focus more and more on the release mechanism until it did the correct action as soon as it was placed in the box. Thorndike called this type of learning "trial, error, and accidental success." Later researchers describe this type of learning as instrumental learning because the correct response is instrumental in providing access to the reward, i.e. the cat has to hit the release mechanism to get its bowl of food.

Computing systems that are modeled to some degree after biological systems are referred to as Biologically Inspired Computing[3] systems. The above example illustrates two of the concepts used in this dissertation. The first is the focused exploration of the environment. In the example above the cat's goal was to get out of the cage and eat the food. At first it had to resort to trial and error to get out of the cage. Once it found a way to get out, in the subsequent trials it focused its efforts on what worked to fulfill its goal instead of continuing to randomly try behaviors. In the work described in this dissertation the robot's initial exploration of the environment is tied to its goals, just as the cat's was. When the robot finds that its goals are being satisfied it continues that action. Since the cat was rewarded only when it got out the cage and to the food, instead of each time it made a swipe of its paw close to the release mechanism, this implies that it had to use its memory of what it did to get out the cage in order to learn. In the learning methods described in this dissertation, memories of exploratory actions are the key to learning correct behaviors. They are used in lieu of requiring the robot to executing

exhaustive trial solutions. Using these memories the robot can learn how to realize its goals in the environment without wearing itself out.

1.1 Goals of This Research

The goal of this research was to develop a technique/system that could produce situation dependent control algorithms for a robot in near real time without requiring that the system physically test multitudes of possible solutions. The main criteria for this system was that it should be able to function without an abundance of a-priori knowledge embedded in either databases, maps, models, or rule systems. The target application was to be one of interest to both the Naval Research Laboratory (NRL) and the robotics community, that being autonomous vehicle/robot formation maneuvering.

1.2 Accomplishments

A generalized approach to learning has been fashioned through the development of a series of algorithms that can be used for construction of behaviors that are able to cope with unstructured environments through adaptation of internal parameters as a result of a goal based supervisory mechanism. Four main algorithms have been developed with two of the four having been tested both in simulation and using mobile robots, and two being tested in simulation. The first two algorithms fall into the class of supervised learning, while the last two are goal directed unsupervised techniques.

The original inspiration for this work was a method called Anytime Learning[4]. This method allows a robot to learn without requiring it to physically execute multitudes of trial solutions. This feature was deemed desirable because of the savings of time and equipment wear and tear that it afforded. The main difference between the presented algorithms and Anytime Learning is that these algorithms do not require the use of a-priori information about the environment. Anytime Learning's functionality is derived through the pairing of a simulation with a robot. Instead of using a simulator, the developed algorithms learn from recent memories of the robots actions within its environment.

A progression of algorithms was developed in order to reach the research goals. A preliminary simulation study was done to show that the basics of the concepts were viable. In this study the groundwork for the Genetic Algorithm (GA) trained Feed Forward Neural Network (FFNN) controller concept was laid. Next a supervised learning technique intended for use as a baseline was developed to show that formation maneuvering could be accomplished using mobile robots with acoustic based relative navigation. The second algorithm automates this process using a goal-based reinforcement learning technique. The third algorithm improves upon the second by not requiring application of a goal function at each time step. This algorithm and the fourth algorithm share features with a class of reinforcement learning algorithms called Q Learning[5].

Q Leaning is an algorithm that learns optimal control strategies through the exploration of its environment. The main strength of Q-Learning is that it allows a robot or agent to

operate in its environment without a-priori knowledge. It does this by learning, accomplished by application of a goal based reward function at each time step in order to optimize actions. The output of the algorithm is a table that recommends actions for each state encountered by the agent. The primary functional difference between Q-Learning and the fourth algorithm developed here is that the Q-Learning table does not preserve information on the proximity of the different states to one another, while the graph structure used in this research does. This information allows features not usually associated with Q-Learning including, tracking of progress towards goals, prediction of sensor values based on actions for self monitoring purposes, selection of alternate routes, and prediction of non-experienced states.

Specific contributions of this research include:

- A GA method of training FFNNs that incorporates finding the transfer function types for the hidden and output layers was developed. This is important because it is not always obvious to the researcher which transfer functions are needed for a given problem.
- Formation maneuvering using FFNN control was shown to work in both simulation and with mobile robots. Most formation maneuvering controllers rely on control laws, behavior schemes, potential fields, or virtual structure approaches. This is important because FFNNs offer a flexible, machine generatable method of control.
- Formation maneuvering using acoustic communication systems was shown to be effective in both simulation and on mobile robots. Most robots that operate in formation use radio, video, or infrared communication systems. While these work well, they are not well suited for the target application in mind, Unmanned Underwater Vehicles.
- A directed method of exploring the environment that is tied to goals was developed. This is important because random explorations, such as those used in Q Learning, can take many time steps to piece together enough data to form a usable operating strategy.
- A method for extending the data collected in exploration was developed based on symmetry of sensors. Using this technique, what is experienced on one side of the robot does not have to occur on the other side of robot for it to be learned.
- A training method based on the clustering of recent memories and sensor prediction was developed that provided excellent results in the target application. This method is an improvement over other methods because it does not require the robot to exhaustively execute all trial solutions and it is unsupervised so no assumptions or rules of thumb are required to provide training examples.
- A graph based method of learning that shares some features with Q Learning was developed. Besides performing well on the target application, the main strengths of this algorithm are that it does not require the robot to exhaustively execute all trial solutions, it is unsupervised, and its graph structure can be used to provide tracking of progress towards goals, prediction of sensor values based on actions, selection of alternate routes, and prediction of non-experienced states.

1.3 Rationale for Doing This Research

Many robots, notably industrial robots, do not have or need the capacity to learn, because their environments do not change. On the other hand, robots that are destined to negotiate changing environments need the capacity to adapt and learn. Nature has many examples of species that can and do adjust their behaviors, whether it is from generation to generation, or by individual learning. Since generation-to-generation learning takes time and populations of individuals, it is not practical or cost effective with the current robot manufacturing techniques. Simulations can help with some of these problems, but simulations have their limitations, especially as the complexity of the environment increases and changes[6]. Given these problems, it is important to find methods that can be used by robots to learn to operate within changing environments without having to execute multitudes of trial solutions in a physical setting, or resort to high fidelity simulations or models.

1.4 Target Application

The target application, robot formation maneuvering, is one that is of interest to both NRL and the robotics community. NRL has an interest in formation maneuvering because of the need to do ocean surveys and area assessments using underwater robots, commonly referred to as Unmanned Underwater Vehicles (UUVs). Teams of these vehicles hold the promise of saving time, money, and reducing risks to humans in adverse environments[7]. By creating formations of UUVs, the sensor footprint (dimensions of the area measured by the vehicles sensors at any instance in time) of the team can be widened. This results in lowering the number of tracks that the formation would need to run in order to survey an area, thus saving time. Making the UUVs job specific can also reduce costs. A team of UUVs may consist of one lead robot with the ability to communicate to the outside world, accurately navigate, and lead a team of less expensive mission specific UUVs. The less expensive UUVs would not carry full fidelity navigation systems, but would carry acoustic sensors adequate for navigating relative to the lead robot and their mission specific sensor package. Using teams of UUVs to do surveys instead of people has the potential of keeping people out of the rigors of a cold, wet and dangerous environment. A key requirement to using a team of UUVs is being able to send the team out to do the job and have it return safely with the data. In order to do this the UUVs will need to work as a team in a formation in a non-communication friendly environment that is constantly changing.

In order to study the UUV team formation maneuvering problem, initial testing was done using computer simulated environments with simulated UUVs. Later, in order to more realistically simulate the UUV problem, testing was then done by simulating UUVs with mobile robots equipped with acoustic communication systems. The mobile robots were used because they are more commonly available, easier to work with, and less expensive than UUVs.

1.5 Overview of the Research

This section provides a brief chronology of the work done towards the research goal of developing a system for the near real time production of situation dependent control algorithms for robots. It provides a general description of the work, ideas, and steps taken during this research. It begins with a description of the base line study done to show that the concept was viable, and continues by describing the work done to develop and test the algorithms that comprise the body of this research.

1.5.1 Formation Maneuvering Using Neural Networks

Initially a baseline study was done in order to determine the feasibility of using neural networks as controllers in formation maneuvering tasks. It was important to determine if neural networks would be sufficient because they more easily lend themselves to machine production than competing symbolic methods such as automated code writing using C or Lisp.

The robots were to use the leader follower paradigm[8]. In this paradigm each robot is assigned a leader. It follows that leader by obtaining information on the leader's location from a centralized tracking system, from the leader itself, by sensing the leader, or some combination of these methods. For testing purposes, the first robot in the line was to be controlled by an operator. The next robot would use its sensors, in this case left and right acoustic pressure sensors (much like our ears) and its control algorithm to follow the leader. This robot would be the leader for the robot behind it and so on. This strategy was chosen because it does not rely on high data rate inter-robot communications, or the use of onboard high fidelity positioning systems. Because communication and position localization is difficult underwater due to the properties of seawater, this passive, low bandwidth, relative navigation system was deemed to be a workable strategy.

For the initial stage of work a formation maneuvering simulation was developed. The simulation provided a sound model and a collision model. The sound model dictated that all robots made unique noises and that the sound dissipated as a function of distance from the source. The collision model was inelastic; it did not take into account any deformation of the robots should a collision occur.

Within the simulation environment neural networks were trained using a variant of a Genetic Algorithm (GA) that not only finds the weights of the neural network but also determines the transfer functions[9]. The advantage of this technique is that the solution type does not need to be known ahead of time by the researcher. Whether it is linear, non-linear, discrete, or some combination of these, the GA finds a combination of weights and transfer functions that works. The GA always caries the best n phenotypes into the next generation and creates the remaining population – m phenotypes from the n best. The fitness function promotes members of the population that can best follow a computer-controlled robot. Said another way, a population of robots each with its own neural network controller was created. Each member of the population was given a chance to follow a computer controlled lead robot. The fitness function scored the individuals on the basis of who could most closely follow the computer controlled lead

robot. The best scoring individuals were transferred to the next generation, and the remaining population was replaced with their children.

Once the controller was "grown" it was installed in simulated robots whose purpose was to carry out formation maneuvering tasks using the leader follower paradigm. Line formations were tested as were binary tree formations. In order to keep robots from colliding with each other, robots that are on the same level as one another are repulsed from each other but attracted to their common leaders.

1.5.2 Mapping the Simulation to Mobile Robots

The work described above clearly demonstrated that the feed forward neural networks were capable of controlling simulated robots in formation maneuvering tasks. The next logical step was to test these ideas in the physical world. For this effort three Active Media Pioneer 2DX robots were equipped with simple acoustic communication systems. The communication systems were comprised of left and right microphones and a speaker.

Using the example of the sound model used in the simulation, a frequency based chirping system was developed. Each leader robot would chirp in an assigned frequency band. A follower robot would measure the acoustic power in its leader's frequency band at each of its microphones, like a pair of ears. In the line formation, all robots but the first and last are both leaders and followers, meaning that the first robot did not need to listen for other robots (it gets its navigation instructions from an operator) and the last robot did not need to chirp (no robots are following it). All other robots both listen and chirp (a chirp is characterized by a noise whose frequency sweeps from one value to another). Chirping was used because constant tones set up standing waves in the lab, making it very difficult to determine the direction of the source. Since there were three robots, two frequencies needed to be selected. Initially the lead robots chirp was from 300 to 500 Hz, and the second robot chirped from 1200 to 1400 Hz. These frequencies were selected in order to avoid difficulties due to harmonics and sound directivity. This strategy is workable, but the amount of usable acoustic bandwidth limits the number of robots that can be in one formation.

Once the acoustic communication system was working well enough to enable amplitude differences to be sensed at the microphones, 3 different control methods were tested. A simple logic system was used as a baseline for initial system testing, followed by a behavior-based system, and finally a neural network system was implemented.

The baseline routine used simple logic and was patterned after Braitenberg machines[10], (light seeking mobile robots). In this routine a follower robot listens for its leader. If the leader's signal is louder on one side or the other the robot heads in that direction. If it's about the same amplitude on either side, the robot goes straight. In order to keep the robot from constantly turning left and right an on-center zone was created. The on-center zone works by letting the robot go straight if the left and right microphone values are close, usually within twenty percent of each other. This routine works well, and its simplicity allows it to deal with various acoustic issues. Using this technique, line

formations were demonstrated. The primary disadvantage of this approach is that, because it looks only at current values, a robot using this method cannot differentiate between heading directly toward the source and heading directly away from the source.

In order to remedy this situation a behavior-based routine with follow, seek, and search behaviors was developed. Follow was the baseline routine described above, seek was similar to the baseline, but without the on-center zone, and search sent the robot into a scan mode that determined the direction of the source. The active behavior is determined by tracking the signal strength at the microphones over time. If the strength is consistently high, the follow behavior is dominant. If it is fluctuating the seek mode takes over. If it is weak, the search behavior becomes active. In simulation, the system surpassed the baseline method because it could solve the front/back ambiguity described earlier. In lab testing with robots the behavior method did not work well. The fluctuations in the acoustic energy, caused by reflections off the walls and floor, at the microphones made behavior selection difficult. The system constantly switched between behaviors, making following a leader a slow process.

A feed forward neural network (FFNN) was also used to control the robots. It is essentially a neural network version of the baseline system, with the exception that the training process determines the size of the on-center zone instead of the operator. The real challenge was to come up with a training process that could be done onboard the robot. Unlike the simulation, it is impractical to let the robot try to follow a preprogrammed leader until it learns how to keep up. In the simulation each robot in the population was given about 200 time steps to follow. In the computer, a population of 50 robots could move through 200 generations in a matter of about 20 seconds. Assuming the real robot would take 20 seconds to make a single run, using the same method would take about 55 hours to train. Even if training could be accomplished in a quarter of the amount of time, the wear and tear on the robots, batteries, chargers and lab walls would be tremendous.

In order to avoid this problem, a supervised training technique was used that paired sensor inputs with locations of the sound source. The robot recorded its sensor inputs along with the relative location of the sound source as it was moved to different locations around the robot. The robot then used the GA to train a FFNN to determine where the sound source was given the sensor inputs. The fitness function promoted FFNNs that correctly determined the location of the sound source. This FFNN was then used to control the robot. It performed well using collected data in simulation and in the robots in the lab. Using this technique line formations were successfully created in the lab.

1.5.3 Modeling of Communications

The goal of the next step was to show that the amplitude difference technique that the listening system relied upon would work in the water. In order to show that the concept was viable for land or water use, several sound measurement studies were undertaken. The tests fell into two categories, amplitude/range measurements and directivity measurements.

The amplitude/range measurements were conducted both indoors and out and over several frequencies and intensities. These measurements were then compared to both a simple medium independent sound spreading model and a more complex model, a version of the U.S. Navy's Parabolic Equation sound model. It was found that the measurements generally followed the modeled results.

As with the amplitude/range measurements, the directivity measurements were performed indoors and out and different frequencies and intensities were used. These tests showed that scattering from the lab walls substantially diminished directivity beyond a range of about four feet. That is, if a person or robot is more than four feet from the sound source, it is very difficult to discern the direction of the source acoustically.

The results of the amplitude/range and directivity tests implied that the work with the simulation and mobile robots should be applicable to formation maneuvering using unmanned underwater vehicles.

1.6 Automation of Learning

Because the robots were intended to operate unattended in distant changing environments, automation of the learning system was required. This section briefly describes the learning algorithms developed in this research. The first algorithm provided basic automation of the learning process and the second improved upon the first by considering more than current sensor values. Both of these algorithms relied on the step- by-step application of the goal parameter. In order to provide more flexibility the third algorithm relaxed this restriction and relied on an unsupervised senor prediction based training system. The final algorithm shared some clustering features with the third algorithm, but instead of using sensor prediction, it relied upon finding the easiest route to a goal state using standard graph algorithms.

1.6.1 Reactive Learning Using Sensor/Action Pairs

The first algorithm was a supervised learning algorithm much like the original human in the loop algorithm previously described. Instead of letting a human tell the robot where the sound source was as in the original algorithm, a supervisory program created a training file from a memory that was created by randomly maneuvering around a sound source. The supervisory program assumed that if the robot had acted to keep up with, or move closer to the sound source, the intensity of the sound would stay the same or increase in a future time step. Using this strategy the human did not have to carry the sound source around the robot and tell the robot its approximate location. Instead the robot used a random walk program and collected a time series of sensor/action data pairs from which the supervisory program created a training file.

An important characteristic of this strategy is that the goal was expressed in terms of sensor values. For a following robot, the goal was to keep the intensity of the sensor values from falling. The assumption was that on average, given a stationary sound

source, or one moving at the same speed as the robot, correct actions would result in sound intensities remaining stabile or increasing. Another feature is that it prepared a training file that was used with the GA training process described earlier. The GA used the training file to create a FFNN that could be used to control the robot.

This technique produced working FFNNs from data collected using the simulator and then from data collected using the mobile robots. In the simulator, the time series of data was collected by driving the robot in random paths towards a sound source and away from it. In the lab, data was collected with a following robot while it followed a leader robot, driven by an operator. The following robot ran the baseline following routine and recorded the sensor/action pairs. The FFNNs were then tested in the simulator and in the robots. Both resulted in effective control in both environments.

1.6.2 Automation of Environment Exploration

In order for the robot to learn its new environment, it had been driven in a random fashion around a sound source. A "good" drive would include many examples in which the robot's actions led to goal optimization. For leader/follower formation maneuvering, the drive needed to contain situations in which the source was to the left of the robot and it had randomly turned towards the source, and the opposite situation in which the source is to the right of the robot, and finally the situation in which the source is nearly in front of the robot and it keeps heading towards it. In early training sessions an operator created a recent memory by recording the time series of sensor/action pairs by "randomly" driving around the source. In keeping with the original idea of creating a system that could learn on its own when it needed too, this "environment exploration process needed to be automated. With this process automatic, the only user input into the system would be the specification of the goal.

The initial versions of the process were completely random. The main problem with these is that they rarely captured enough usable information before they steered the robot so far away from the source that there was no appreciable amplitude differences at the sensors, meaning there was little information available to make a decision from or from which to learn. To remedy this a "random but purposeful" approach was taken. In this approach the robot would remain going whatever direction it was already going as long as the goal parameter was being satisfied, but as soon as it was not it would change course randomly. The controller had no knowledge of how to act based on its sensor inputs, it only knew that what it was currently doing was either satisfying its goals or not, and that if its goals were not being satisfied it should do something else. This routine had the effect of keeping the robot hovering around the source in a butterfly like manner. This controller could effectively provide examples that could be learned from.

1.6.3 Non-Reactive Neural Networks: Taking Past Events into Consideration

Initially, the routines and neural networks that worked in both the simulation and on the mobile robots were strictly reactive in nature. That is, they only considered current sensor readings. The behavior routine considered past values, but because of acoustic issues its performance in the lab was inadequate. In order to solve the problem of the robot heading directly away from the sound source and not knowing it, consideration of past values is a requirement. Without looking at past sensor values, or some parameter that is a reflection of them, there is no way that the robot can tell if is currently doing better or worse than it was in the near past. Without this information a decision to turn around cannot be made.

Both FFNN and recurrent neural networks can consider past values. A recurrent neural network is more flexible than a FFNN because it can consider an arbitrary number of past values but its structure is more complex than the FFNN. A dynamic FFNN can only consider a fixed number of past values, but its structure is straight forward. For this work the FFNN was chosen because of its simplicity. The idea at the time was to see if a FFNN network could solve the problem. If it could not, recurrent networks were to be explored.

For these tests a "principle parts" routine was developed to provide contiguous runs of sensor value/action pairs for the FFNNs. In the earlier routines, because they only considered one value at time, the data points could be in any order, or data could be filtered so that only values that optimized the goal by some percentage were used.

Because ordering was now important, the principle parts routine examined time series data collected from the random but purposeful controller and separated it into four categories. The categories were differentiated by the way that the characteristics of the goal parameter behaved. For the leader follower problem the goal parameter is overall sensor intensity. The categories were positive, negative, equal, and unknown. In the positive category the goal parameter would generally be gaining strength, in the negative category it is loosing strength, in the equal category it is not changing, and finally, in the unknown category it is usually fluctuating, i.e. it does not fall in any of the previous categories.

Initially training was done only with the positive category. Eventually, all categories except the unknown category were used in training. Equal was treated like the positive category, but the negative category was handled differently. To train with a negative example, the network was encouraged to take the same action that used in a positive or equal run that whose run time was adjacent to or overlapped the negative run, if such a positive or equal run existed. Using this strategy a FFNN was developed that could under certain conditions solve the problem of the robot heading directly away from the sound source and not knowing it. The conditions were that the sound source had to be moving quick enough in relation to the robot for there to be a gradient present in the number of past values presented to the neural network. So if the source were moved directly towards the front of the robot slowly the robot could be fooled. When the source passed

over the robot and started moving away, the robot would not be able to detect it. But if it were moved more quickly the robot would track the source by turning around and aligning its sensors toward it. These tests were carried out in simulation.

In order to prompt the neural network to learn how to do all actions the fitness function was set to encourage the correct actions as a percentage instead of directly counting correct actions. By using percentages, the scores of an action that has more examples in the recent memory than another cannot overshadow actions that are less represented. Another benefit of this method is that the importance of actions can be weighted in the fitness function.

1.6.4 Unsupervised Learning Based on Sensor Value Prediction

The algorithms discussed earlier that ran on the robots were supervised algorithms. That is, for each sensor input in the training set, either a human or a supervisory program indicated the correct action. This technique is much different than the unsupervised technique used in the original formation maneuvering simulator. Recall that the fitness test for that simulator was based on a follower robot's average distance from a computer controlled leader robot. Here the fitness function measured the results at the end of the run of a robot. The robots that remained closest to the leader were promoted. Instead of trying to teach the robots what actions were best based on checking a goal parameter at each step, the robots in conjunction with the GA found the senor/action pairs that combined to satisfy the overall goal. This technique works well if the goal is known and it is hard to discern the correct action at each instance in time, but in the simulator it did require that multitudes of runs be made, which is not practical for a physical robot. A method was needed that could make use of the unsupervised strategy but avoid repeated physical execution with the robot.

This was accomplished by using a clustering technique to break down the recent memory into a library of overlapping experiences. For example, following a leader robot could be broken down into several overlapping experiences that include making a left turn when the leader is just to the left, making a left turn when the leader is further to the left, making a right turn when the leader is to the right, etc. Learning is accomplished by asking a candidate controller what action it would take if it were in the situation described by an experience. The controllers that answer in ways that consistently lead the robot towards its goals are promoted over those that don't.

The mechanics of this process entail forming a new experience from the answer provided by the controller and the experience it was asked about. This new experience is compared to each experience in the library of overlapping experiences. From this comparison the closest match to the new experience is located in the library of experiences. Next, the original experience that the controller was asked about and the closest match to the newly formed experience are compared to each other. If the closest match is more near to the goal than the original experience, the controller scores points. Using this strategy a GA is used to produce a FFNN controller. Chapter 6 provides a detailed explanation of this process.

The algorithm was tested in both the leader/follower simulation and using lab data collected by the mobile robots in leader/follower maneuvering situations. In both cases working controllers were created. It was noticed that sometimes a controller was produced that could not do all functions. It was determined that the recent memories being used to learn did not contain an equal distribution of experiences to learn from, thus a controller could score well, but not be able to do all required functions.

In order to help mitigate this problem, a sensor mirroring technique was developed. This technique is based on symmetry of the robots sensors. The general idea is as follows: if an action creates an increased stimulus on one side of the robot, the opposite action will create a similar stimulus on the other side of robot. Since the robots have only two microphones and were restricted to 3 actions, left, straight, and right, this idea was easy to implement. Using this technique, a memory that contained predominately left and straight turns could still be used to teach the robot when to make right turns.

The sensor mirroring idea is a broad-brush approach to solving the problem of incomplete memories, while the percentage based fitness function works to even out the distribution of different situations contained in the recent memory. The next paragraphs describe a technique developed to provide a structured method of analyzing the solution space, with one of its benefits being a more focused method of sensor mirroring.

1.6.5 Graph Based Learning

All the algorithms discussed have used either a supervised training set or recent memories to produce a FFNN. Once that FFNN is produced the recent memory is not used. The final method provides the ability to use the recent memory in conjunction with a FFNN to control the robot, monitor its progress towards its goals, provide alternate methods of goal realization, predict sensor values based on current actions and sensor values, and finally it provides a structure from which to infer the existence of non-explored areas of the solution space. As in the technique discussed in the previous section, clustering is used to separate a recent memory into a sequence of experiences. By examining this sequence, questions like "Which experience usually follows this experience?" can be answered. From this type of analysis, a map of the connectivity of the experiences is formed. This map, called a directed graph, can be used not only to learn, but also provides additional features described in the following paragraphs.

By defining one or more experiences as goal states, learning based on the shortest path from undesirable states to the goal states can be accomplished. This learning algorithm, shares some attributes with Q Learning in that it has similar optimization functions and they are both dynamic programming methods[11]. Using this technique a FFNN was successfully tested in simulation. It was created from a memory collected by the automated recent memory collection scheme described earlier. Learning was demonstrated using this approach in this work. This approach offers the potential of many advanced capabilities beyond what was demonstrated and is left for future work, described by the following bullets.

- The graph can be used as part of an observer module. As the robot is using a FFNN to operate in its environment its current sensor readings can be compared to the experience map thereby tracking the current state of the robots progress. When it remains in a "bad experience" for too long the observer can take action such as notify an operator, or kick off an escape process.

- Shortest path graph algorithms can be used on the experience map to find the quickest route from bad experiences to good experiences. These routes can be used to get the robot out of local minima in which the FFNN robot controller may get stuck. Also, by relabeling graph edges alternate routes to goal states may be found.

- By analyzing the graph, the existence of nodes that are not represented in the recent memory can be inferred by using techniques such as symmetry, interpolation, and extrapolation. By creating these nodes, the portion of the solution space that they represent would not have to be directly experienced. This would represent a rudimentary form of reasoning based on known facts using simple rules. If these nodes were included in path calculations to stabile states, they can be used in training FFNN from paths.

1.7 Organization of the Dissertation

The next three chapters provide a literature review and cover the applied portion of this dissertation. Chapter 2 provides a literature review. The three topics covered are robot formation maneuvering, robot architectures, and learning systems used in robots. General information is provided along with more specific information that had direct influences on this research. Chapter 3 details the baseline study done to show that the idea of creating a FFNN on the fly would work for the target application, robot formation maneuvering. It includes information on the relative navigation approach to implementing the leader/follower paradigm. Along with details of the multi-robot simulator developed for this section of the research, it includes a discussion of the modified GA used throughout this work. Chapter 4 discusses the transition of the work detailed in Chapter 3 to mobile robots in a lab setting. Topics such as the acoustic communication system, its relevance to UUVs, and basic control algorithms are covered.

Chapters 5 and 6 provide a theoretical foundation, implementation details, and experimental results for the 4 major learning algorithms described in this dissertation. Chapter 5 provides details on the supervised algorithms used to create the controllers for both the simulated and mobile robots discussed in the previous chapter. This chapter discusses such issues as why learning from recent memory is used, reactive vs. non-reactive control, sensor mirroring, and principle-parts learning. Chapter 6 continues the learning discussion begun in chapter 5, but changes topics to cover a type of non-supervised learning based on clustering and sensor prediction. The rational for this approach is discussed along with detailed conceptual and implementation information. The discussion is continued with a description of a graph based learning method that uses the structure of a connected graph to provide an unsupervised method of learning.

The final chapter, chapter 7, provides conclusions, discusses issues, describes improvements that can be done with the existing algorithms and closes with a section on future work.

Chapter 2. Literature Review

This chapter provides background material for the topics of: UUV teams and how they are related to the area of robot formation maneuvering, robot architectures, learning systems used in robots, and acoustically guided robots. General concepts are presented along with more specific information that has direct relevance to this research. Although this research is not specifically about formation maneuvering or UUVs, some general knowledge of these topics is vital to the development and testing of the learning systems destined for them. The same can be said for robot architectures; this work is not intended to advance that area, but since the learning algorithms studied here are to be housed in a robot architecture, general knowledge of this topic is essential. Also, background material and specific examples of learning systems that this research is either similar to in function or operation are provided. Finally, the acoustically guided robots section provides a short discussion of 3 systems that use acoustics to guide robots to a sound source or to orient it in the direction of the perceived sound.

2.1 Unmanned Underwater Vehicle Teams and Formation Maneuvering

NRL has an interest in formation maneuvering using UUVs because of the need to do ocean surveys and area assessments. Currently survey ships and divers do these tasks. It is hoped that automated data collection systems consisting of teams of UUVs will lower the cost of the equipment, save time, and reduce the risk to divers working in adverse conditions caused by low water temperatures and currents. UUV teams hold the promise of lowering costs, saving time, and reducing risks, by utilizing formations in which the sensor footprint can be regulated by the number of mission specific UUVs. For example, by using a formation that results in a wider sensor footprint the formation can traverse the survey area in fewer tracks, resulting in a savings of time. UUVs that are mission specific can be used to do the data collection task, while a single higher cost lead UUV can be responsible for overall team navigation and communications with the host vessel. In order for this scheme to work the UUVs need to work as a team in a formation in the ocean environment. This section provides a description of the steps involved in what is expected to be a typical mission for a team of UUVs[9]. Following that is a literature review of formation maneuvering. Four major maneuvering paradigms and their applicability to this work are described.

2.1.1 Typical Unmanned Underwater Vehicle Mission

A conceptual mission involving multiple UUVs will have many distinct phases. Initially, the UUVs will be onboard their host vessel(s). Depending on the size of the UUVs involved in the mission and the mission goals, there may be more that one host vessel deploying UUVs. Once sea prepped the UUVs will be launched and they will then maneuver into a transit formation and travel to the area of interest. The current assumption is that there will be at least one vehicle that has an accurate positioning system on board and that the others will rely on vessel-relative positioning. Upon getting to the area of interest, the UUVs will change into mission specific formations and execute

their respective mission related goals. When the mission is complete they will move back into a transit formation and return to their host vessel(s) and download their data. Figure 2.1 below illustrates the phases of a three UUV mission.

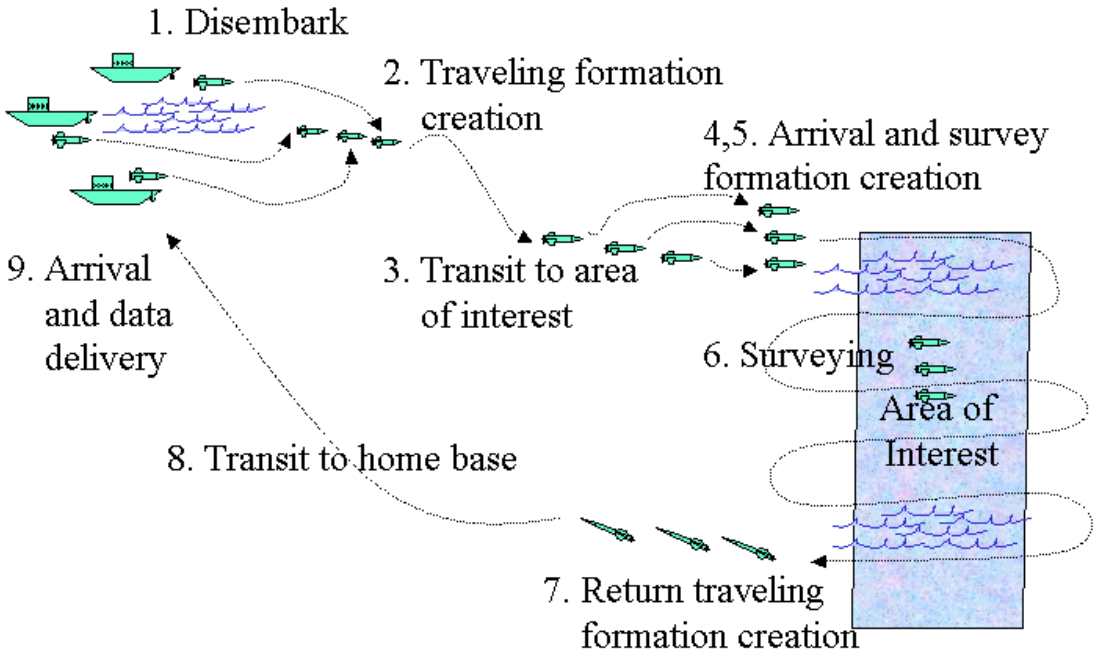

Figure 2.1. The phases of a conceptual multi-UUV mission. In this mission three UUVs are launched from three separate ships (1). They make a simple line formation, travel to an area of interest (2, 3) and upon arrival switch into a three abreast formation (4, 5) to do their survey (6). When the survey is complete they switch back to the line formation (7) and return to their ships (8, 9).

As can be seen in the illustration in figure 2.1 and by the description of the typical UUV mission, creation of formations and formation maneuvering is key to the mission's success. The next section has a review of formation maneuvering as it applies to mobile robots. As stated in chapter 1, mobile robots were used to model UUVs in this work because they are more commonly available, easier to work with, and less expensive than UUVs. The four major maneuvering paradigms are described and there is discussion of similarities and differences with the work in this dissertation.

2.1.2 Formation Maneuvering

Coordinating multiple autonomous vehicles moving in formation is important from a robotic/UUV standpoint in that it can be a key part of getting a team of robots to work together[12]. As referred to earlier, one use may be to create a formation of UUVs and travel to a destination of interest in order to collect measurements of depth, physical water properties or to look for hazards. Using a formation helps the team members track other team members, helping to make sure that none of the UUVs gets lost along the way. It also augments communication by reducing the distance that a team member will have to transmit a message. Formation maneuvering based on inter-vessel positioning and

navigation has distinct advantages in that it can reduce or eliminate the requirement for pre-deployed positioning systems. Another use of a formation is to increase the sensor footprint in searching and surveying tasks.

Existing works on formation control fall into one of the following 4 categories:

1. Leader-follower approach: each vehicle (except a special leader vehicle) follows one or more designated leader vehicles with required relative position and orientation. The group in University of Pennsylvania has done extensive work in this leader-follower paradigm[8 13 14 15]. The stability of the control laws developed for vehicles that follow one or two leaders has been established[8].

2. The behavior-based approach[16] : Motor schemas (algorithms) are designed for each of the basic behaviors (such as keeping formation, avoiding obstacles, etc.), and the control command (heading and speed of the vehicle) is obtained by a weighted combination of the basic behaviors.

3. The potential field approach[17 18 19] : this approach associates an artificial potential field with a group of vehicles. The potential and the interaction force between a vehicle and its neighbor vehicle are dependent on the distance between these two vehicles. The artificial potentials should be designed in such a way that the desired group configurations are obtained at a global minimum of the potential function. The movement of the vehicles of the group should be toward minimizing the potential.

4. Virtual structure approach[12] : consider the group of vehicles as forming a rigid structure, and move the group along a trajectory with control laws that minimize the deviation of each vehicle from the desired position in the virtual structure. This is different from leader follower in that the vehicles each try to maintain their relative position within a group of vehicles. In this way it avoids single failure points that would exist in some leader-follower formations such as a line.

Compared with existing works about formation control, this work has some distinctive features: it makes use of a machine learning technique to learn the control laws to move into (acquire) formation, and to keep (follow) formation. The acquire networks were trained using a fitness function that emphasized a high degree of movement as well as being able to hold station, while the follow network's sole purpose was to keep the distance between leader and follower at a minimum. The follow network was used in the majority of this work. In one of the most relevant works, a genetic algorithm is used to evolve neural network controllers for simulated "prey" creatures to learn a herding behavior protecting against predators[20]. However, that work does not address the issue of forming a particular geometric shape (line, tree, etc.).

Another key difference is the inter-vehicle communication scheme. In our scheme we navigate relative to an assigned leader using a frequency multiplexed chirping scheme and acoustic sensors. The group at the University of Reading[21] has concentrated on both

non-adaptive and adaptive flocking behavior with mobile robots using logic based on that of Reynolds's "boids"[22]. Their robots are using ultrasonic sonars for obstacle avoidance and frequency multiplexed infrared light for inter-robot communications.

Other more common methods include the use of camera based, laser based or GPS positioning systems coupled with high-bandwidth communication networks which inform each vehicle in the formation of its position and the position of its leader or leaders. While these approaches have merit and in general work well they are not suited to the ocean environment because the physical properties of seawater do not lend themselves to transmission or reception of high bandwidth radio frequency communication signals, or passive sensing systems based on vision.

2.2 Robot Architectures

Robot architectures are important to this work because the learning system will ultimately reside in the control architecture of either a physical or simulated robot. It will interface with other parts of robots control system such as the observer and the execution system. While it is not the focus of this research, the way that these various parts interact with one another and the outside world is important. This section provides a brief discussion of robot architectures, including some historical information, and a more detailed discussion of Three Layer Architectures is included in the following sections.

2.2.1 Sense-Plan-Act

According to Gat, the predominant view of the artificial intelligence (AI) community[23] before the introduction of Brooks' Subsumption Architecture[24] was that the control system for an autonomous mobile robot should be decomposed into three parts: a sensing system, a planning system, and an action system. This is commonly referred to as sense-plan-act (SPA). The sensing system took raw sensor data, processed it and inserted it into a world model. The planning system uses the world model and goals to generate a plan to achieve the goals. This plan is submitted to an action system which in-turn converts the plan to actions, which are then carried out by the robots actuators.

Significant features of SPA are:

- Information flows from the sensors to the world model/planning system and then to the actuators. It never flows in the reverse direction. SPA is sequential and unidirectional in nature.
- Execution of a plan in SPA is similar to executing a program. Both are made of primitives combined with control statements.
- The intelligence of the system resides in the planning module. This module generates plans based on goals and current states. To quote Gat: "Research efforts through about 1985 focused almost exclusively on planning and world modeling."

While SPA definitely has had its successes, the difficulty involved in world modeling and planning limited its success. Even with perfect world modeling, noise from sensors could easily cause the information that the sensors return to become out of sync with the world

model. This in turn would cause errant planning and incorrect actions. Realizing these limitations, researchers began to look in different directions. Brooks was one of the first to come up with an alternative approach.

2.2.2 Behavior Based (Brooks' Subsumption)

MIT's AI Lab has done extensive work in the field of "Behavior Based" robotics. The lab is credited with creating the Subsumption Architecture, Behavior Language (BL) and a cast of mobile robots that operate successfully in unstructured dynamic environments.

The Subsumption Architecture is a method for controlling mobile robots that differs markedly from centrally controlled, top down approach, classical AI methods (SPA). It can be described as a hierarchical, asynchronous, distributed system whose components produce behaviors. The interactions of the behavioral components in the subsumption architecture produce the system functionality. In general, sensor inputs are available throughout the system and thus sensor input and actuator actions are closely coupled. The behavioral components are set up in a hierarchy of layers, in which higher layers can inhibit or subsume lower layers, thus the name Subsumption Architecture. Figure 2.2 illustrates the differing approaches of classical symbolic AI (SPA) and the Subsumption Architecture.

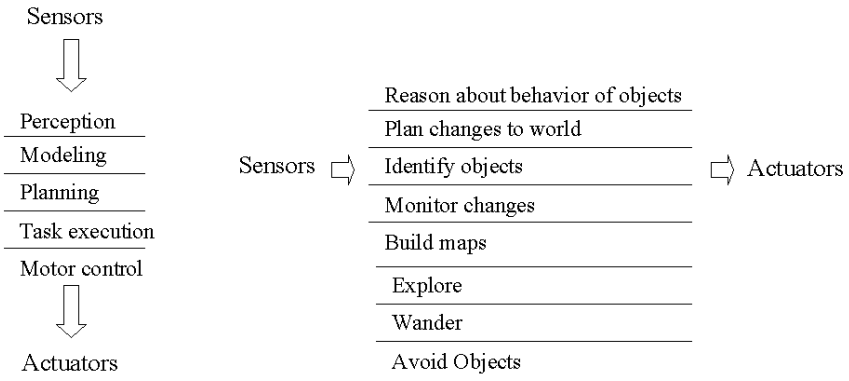

Figure 2.2. The differences between the Classical AI approach, figure reproduced from[24], SPA and that of Subsumption are illustrated in this figure. SPA is much more of a top down, centrally controlled approach while Subsumption takes a bottom up distributed approach.

Gat explains that SPA runs into problems when its internal state loses sync with reality. This is a result of the environment changing while the machine is calculating its next move. As shown in figure 2.2, SPA is a sequential process. If the time between sensing and acting is taken up by some long-term processes, then changes in the environment will cause the world models states to be different than what is actually happening in the environment, a loss of sync. Another problem associated with SPA is that when a step in a plan fails, subsequent steps are out of context.

The subsumption (reactive) solution to these problems is to minimize tracking of environmental conditions using internal variables. Gat says that the reasoning behind

subsumption is that if there are no internal states, then it (the robot) cannot lose sync with the world. He goes on to point out that this idea is nice until the sensor readings are inconsistent. He used the example of a robot that could not see a wall because of sensor noise. If it had detected the wall earlier, but now could not see it, it may have been a good idea to remember that a wall had been near by earlier. But because subsumption tries to minimize the use of states, the robot may run head long into a wall that it had previously detected.

Both SPA and subsumption have their pros and cons. The following bullets summarize the situation:

- SPA runs into problems when its internal states become out of sync with its environment. The common cause of this problem is due to its sequential nature and long execution times of its world models and planners.
- If a step in an SPA generated plan fails, the subsequent steps are executed out of context.
- Subsumption avoids getting out of sync by minimizing the use of internal states (no world models or planners here) and by executing its "behaviors" in parallel.
- Because it relies heavily on sensors and has no models, subsumption is highly influenced by sensor noise.

Understanding of these two semi-opposing viewpoints, SPA and subsumption, is important because some more recent developments in robot architectures make use of them both. Three layer architectures are a prime example of this. They attempt to exploit the benefits of each while minimizing the drawbacks[23].

2.2.3 Three Layer Architectures

Three layer architectures try to manage these problems by organizing algorithms based on how they handle states[23]. Algorithms that are reactive, stateless sensor based algorithms reside in the control layer. These are like behaviors in Brooks' architecture. Algorithms that use states that track the past make up the sequence layer. Those that deal with the future inhabit the deliberator. The following sections provide more detail about each of the layers. Figure 2.3 below shows a conceptual view of the Three-Layer Architecture. The following sections discuss each of the layers.

In the control layer one or more control processes are running simultaneously. These processes consists of tightly coupled sensor/actuator feedback loops. Much like subsumption, these processes usually implement a primitive behavior such as wall

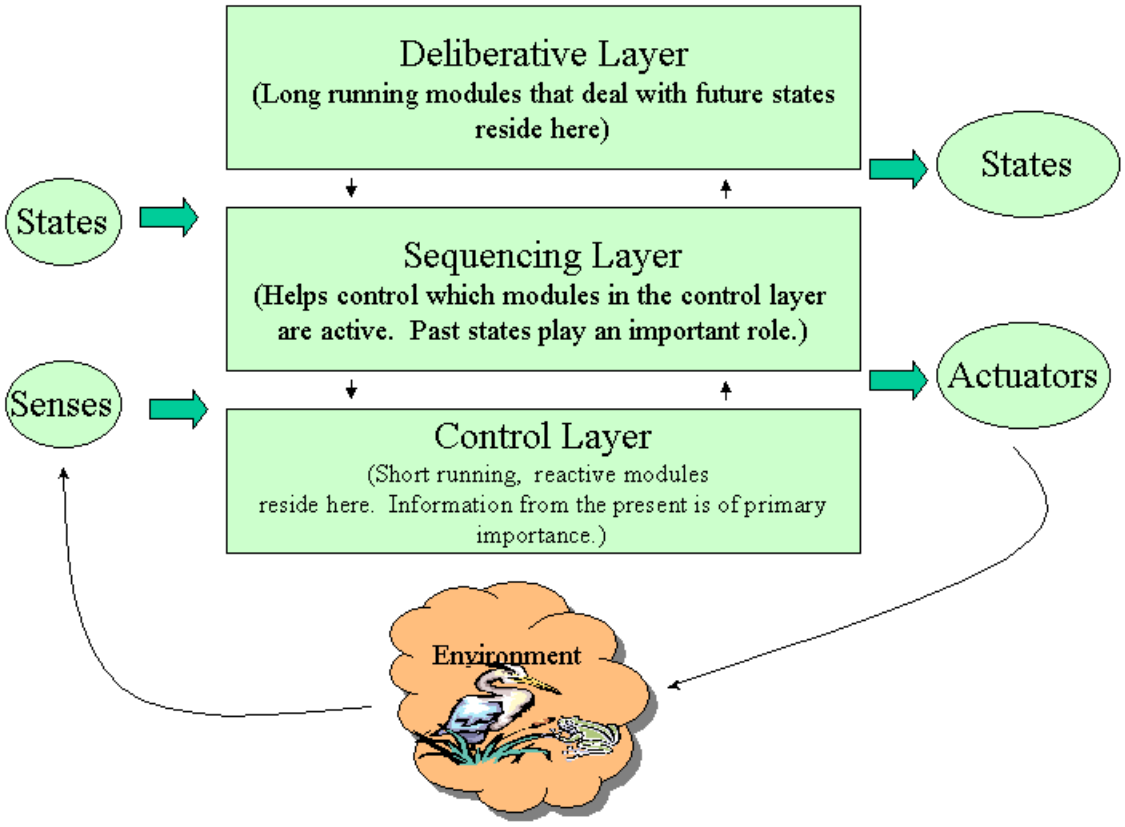

Figure 2.3. The conceptual structure of the Three Layer Architecture. The Deliberative layer has long running modules that resemble planning and modeling done in SPA. The Control Layer has short running, present state modules that resemble Subsumption. The Sequencing Layer acts as an arbitrator and interface between the Deliberative Layer and the Control Layer.

following, avoiding collisions, etc. Listed below are the architectural constraints for controller level processes.

1. A single iteration of a controller process should be of constant-bounded time and space complexity.
2. When an algorithm fails it should be able to detect its failure. This allows other parts of the system to take corrective action if need be.
3. The use of internal state should be minimized. Most of these algorithms are reactive in nature. In this context the term reactive refers to a process that considers only the current state. It does not consider past trends or change what it will do in anticipation of future events.
4. The controller process should not introduce discontinuities in a behavior. In other words, a behavior should do one thing; it is the job of the sequencer to manage transitions between behaviors.

It is the sequencer's job to select which behavior is appropriate for a given set of circumstances. In this way a robot is able to do useful things like travel from point A to

point B while avoiding any objects along the way. In the original subsumption the action of high-level behaviors subsuming lower level behaviors is analogous to a non-adaptive sequencer that does not consider any states.

Approaches to designing sequencers are briefly described below:

- Enumeration - All states that the robot can be in are identified and the correct state for each given task is pre-computed. The trouble is that it is not possible to always know the current state, especially if unexpected events occur.
- Conditional Sequencing – Quoting Gat; "Conditional sequencing provides a computational framework for encoding the sort of procedural knowledge contained in instructions. It differs from traditional plan execution in that the control constructs for composing primitives are not limited to the partial ordering, conditionals, and loops used to build SPA plans. Conditional sequencing systems include constructs for responding to contingencies and managing multiple parallel interacting tasks."[23] One way that these systems are implemented is by using a purpose built language that provides for these constructs.

The job of the sequencer is to select which action(s) are needed at the current time. Its operation rate should be complementary to the rate dictated by the environment and the behaviors in the controller. For example, in a mobile robot it should be able to select the output of the avoid obstacle behavior without pondering what to do for so long that a collision occurs.

Long running algorithms reside in the deliberator. A good example may be a path-planning algorithm. The key architectural feature of the deliberator is that several behavioral transitions can occur in the time that an algorithm in the deliberator starts and ends. That is, several cycles of the control and sequence layer can occur while the deliberator routines are executing. The deliberator can send plans to the sequencer or it can serve plans up to the sequencer at the sequencers request.

Three layer architectures have been implemented using several different approaches. In general the behaviors in the controller are hand crafted (in C, Lisp, or a purpose built language) and adhere to the previously outlined architectural constraints. The sequencer routines represent plans for the robot. They can make extensive use of state variables such as position and locations of objects of interest. A sequencer plan can be written in a language like Lisp and contains logic and calls to controller level routines for action events and can contain calls to deliberator level routines. In some systems the deliberator routines are active at the request of the sequencer, and in others the deliberator directs the sequencer. Iyengar et al[25] report on a system that handled both real time tasks and planning tasks by combining the power of the CLIPS production based AI language with a multi-processor control architecture.

The significance of Gat's discussion is the way that the states are handled in the different layers and the architectural conventions imposed on those layers. While the focus of this

work is automatic behavior generation, some of Gats handling of states and architectural guidelines were followed.

2.3 Robot Learning Systems

This section discusses recent approaches to the learning problem. The first method of learning, learning under subsumption, is included because subsumption was so influential in the mid to late 80's on through the 90's. The article discussed is important because it illustrates a learning method that worked well, but it required the robot to make many trial movements. The second method discussed, Anytime Learning, is a method that helps the robot avoid making those trial motions, but it requires a simulation system, exactly what advocates of subsumption argued against. The next method, detailed in a paper by Jirenhed, Hesslow, and Ziemke[26], describes a neural network method in which an "Inner World" or computational imagination was used to create obstacle avoidance routines for a mobile robot. This paper is discussed because their methods and results are very similar to this research's early results obtained by using a direct solution to the Weiner Hoff equations. The final method discussed, Q-Learning, does not require a world model and it allows the robot to learn without having to specifically test and retest movements in order to learn.

2.3.1 Subsumption - Learning Arbitration

Learning under the Subsumption Architecture is primarily a matter of conflict resolution among the various behaviors. That is, deciding which behaviors should be active at any particular instance in time. Historically the developers resolve these conflicts at compile time or by using a description of the desired behavior selection. In both cases, the resulting behavior hierarchy is static.

In 1990 Maes and Brooks[27] described an algorithm which allows a behavior based robot to learn based on positive and negative feedback, thus eliminating the need for the programmer to solve all the potential walking problems ahead of time. The end goal was to control overall robot functionality by selecting the appropriate behaviors from a "library" of behaviors, defining the connections to the sensors and actuators, defining the positive and negative feedback functions, and then letting the behaviors learn from experience when to become active.

As with all behavior based robots the algorithm is mostly distributed[27]. Each behavior tries to find out, based on feedback, when it is appropriate for it to be active. That is, in what situations is a particular behavior's operation relevant? Also, what are the conditions in which it's operation becomes reliable? Said another way, when is the behavior's operation consistently associated with positive feedback? Based on feedback, each behavior is able to determine its own relevance and reliability in the context of a given situation.

Using this algorithm the six legged robot "Genghis" successfully learned to walk using swing forward behaviors for each leg, avoiding negative feedback produced when it fell on its belly, and maximizing positive feedback generated when it moved forward. Genghis learned the tripod gait, keeping three legs on the ground at any one time: the

middle leg on one side and the front and back leg on the other side. This is not a trivial task, neither the negative nor the positive feedback is associated with any single behavior, but with the coordinating of the behaviors in a distributed system.

2.3.2 Anytime Learning

In the previous discussions the idea of learning has not been mentioned except in Maes's algorithm for learning to coordinate behaviors. Recalling the thrust of the earlier discussion, this method concentrates on learning when reactive behaviors are appropriate. The three-layer architecture provides a methodology that helps to arrange the system based on how state information is handled. In Gat's version, it contains handcrafted behaviors, plans, and tools such as path planners for aiding in plan completion, but it does not specifically mention learning of new behaviors or plans. The next paragraphs detail an idea called Anytime Learning[4]. It has some similarities to the state separation theme of the three-layer architecture.

Anytime Learning describes a particular way in which a learning system can be integrated with an execution system. The basic idea is that the execution system and the learning system run together continuously. The learning system contains a simulation of the environment in which the execution module operates. Its job is to continuously test strategies in the simulation so it can identify and develop improved strategies that it can then provide the execution module. Meanwhile, the execution module is operating in the environment collecting relevant parameters about performance of the strategies and any changes in the operating environment. This information is used to improve the fidelity of the simulation. Figure 2.4 below shows a conceptual diagram depicting the major components of Anytime Learning and their relationship to one another.

Parker used a variant of Anytime Learning, punctuated Anytime Learning, to develop gaits for hexapod robots and to evolve team behavior in box pushing tasks[28]. The term punctuated means that the learning system is not in the robot with the execution system. Since it is not close at hand, it does not have immediate access to sensor data, so the simulation in the learning system is not continually kept in sync with the environment. Punctuated Anytime Learning is a modification that compensates for the lack of feedback, precise sensors, and high computational power in the robot.

Fitness Biasing and Co-Evolution are the methods that Parker uses to update the offline simulator that learns using a genetic algorithm (GA)[28]. In Fitness Biasing, after n generations are run in the simulator, the entire population of solutions is tested in the real environment. The results of simulator and the real environment are used to bias the simulator so that its results are more in tune with those of the real environment. In Co-Evolution, the simulator itself is evolved along with the solution[28]. Its parameters are encoded in a GA and the fitness is based on how close the results and the real results match the physical situation.

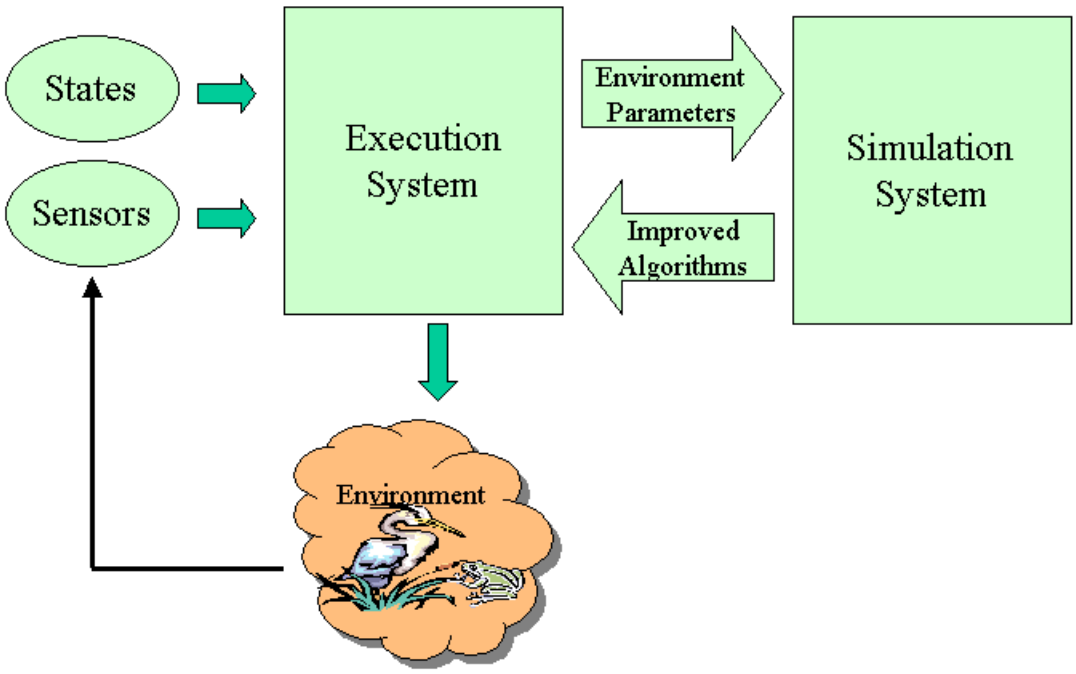

Figure 2.4. The basic components and their relationship to one another in a typical Anytime Learning System. The execution system accepts inputs from states and sensors, makes decisions and controls actuators, which in turn influences the environment. Meanwhile it sends environmental parameters to the simulation system, which uses them to improve the fidelity of the simulation. The simulator takes advantage of the computers speed and finds improved control strategies, which it in turn provides to the execution system.

In this work, the functionality that Anytime Learning provides, that is the ability to come up with possible alternative solutions that have a reasonable chance of being better than the existing solution, is key. As stated in the introduction, the proposed work differs from Anytime Learning in that instead of using a simulator as the "imagination" of the robot, a neural network based sensor prediction/learning system is developed.

2.3.4 Learning and Prediction Using Neural Networks

This section provides a short review of a paper by Jirenhed, Hesslow, and Ziemke[26]. Their goal, similar to the goal of the research described in this dissertation, was to create a method for robots to learn behaviors without having to physically execute them, and without using traditional AI techniques or simulators such as those described in the previous section. This paper is discussed because their methods and results are very similar to this author's early results obtained by using a direct solution to the Weiner Hoff equations.

Jirenhed replaced traditional AI symbol manipulation and simulators with a sensor prediction recurrent neural network in order to train mobile robots to avoid obstacles. His 2nd Hypothesis was: "The organism can simulate long chains of actions and

perceptions without any external input. This simulated interaction world will inevitably appear as an inner world".

In order to test this hypothesis he ran three experiments using simulation and mobile robots. First, using a GA to train a FFNN of fixed topology he trained the robot to avoid obstacles. Next the robot was taught to predict sensor values. The sensor values for time frame(current time + 1) were predicted using the current time frames sensor inputs and the context of that time frames input. This resulted in a rudimentary sensor prediction capability. In the third experiment the predicted sensor values were used as the input in order to create an internal simulation. For this experiment he reports no reliable behavior for extended periods of time, that is, turns were not handled correctly. He postulated that the neural network had too few hidden nodes for making long sequences of sensor predictions, or that the limited sensor range of the robots made the prediction task too difficult.

As stated earlier, these results are very similar to work in which this author predicted sensor values based on previous actions and sensor values using a neural network[29]. In this work, the weights were found by using a direct solution of the Wiener-Hopf equations. Predictions were good for the first 20 to 30 percent of the path and then they degraded.

2.3.5 Q Learning

Q Learning falls into a class of algorithms called Reinforcement Learning[5]. In general Reinforcement Learning algorithms provide an agent with the ability to reach its goal by providing it with a means to select optimal actions.

Sutton and Barto summarize the ideas behind this type of learning in the first chapter of their book *Reinforcement Learning: An Introduction*. They say:

"Reinforcement learning is learning what to do--how to map situations to actions--so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics--trial-and-error search and delayed reward--are the two most important distinguishing features of reinforcement learning.[30]"

The agent is assumed to be able to acquire information about its environment, i.e. sense its state, and its actions are assumed to change its state in a measurable way. Looking at the diagram in figure 2.5 taken from Mitchell[5], if an agent were in the block to the left of the goal state (the state marked **G**) it would need to be able to sense that it were in this state. A move to the right would always send it to the goal state.

The idea of delayed reward is key to Q learning. Delayed reward is used because the supervisor or trainer only knows when the agent achieves success, which is defined by reaching the goal state. The idea behind delayed reward is that the total available reward

for reaching the goal state is discounted by the number of steps that it takes to reach the goal state. Looking at figure 2.5, there are six blocks in the diagram; each one corresponds to a state. The only time an immediate reward is given is when the agent moves from a state adjacent to **G** into **G**. When this occurs it receives an immediate reward of 100 points. All other transitions result in an immediate reward of 0 points.

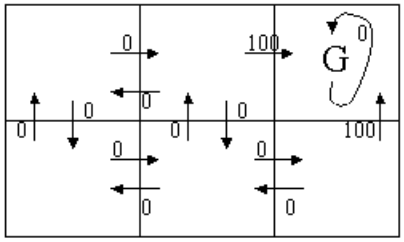

Figure 2.5. A simple 6 state world. Each square represents a state, and each arrow is an action. The numbers next to the arrows are the immediate reward values. As can be seen from the figure, only the arrows leading directly into the goal state **G** return immediate reward[5].

The discounted rewards for each state and each action are calculated using the Q function. The Q function is defined as the sum of the immediate reward available at the current state plus the discounted reward. Its equation is shown below.

s - current state
s' - the state in the set of states adjacent to state s that contains a'. (s', a') is the optimal state/action combination adjacent to s.
a-current action
a' – action that returns the greatest reward at state s'
d-amount that rewards are discounted, range is a real number between 0 to 1
r – immediate reward available for current state.

$$Q(s, a) = r + d * \mathbf{max}\ Q(s', a')$$

The **max** function in the equation indicates that the state, action pair (s', a'), adjacent to (s, a), that delivers the maximum reward is selected.

An example calculation illustrates the process. Consider the state that is at the lower leftmost position in the diagram. The reward for this state is arrived in the following manner. For this example the discount value, d, is set to 0.9.

Working from the state s = lower left, towards **G**

Q(lower left, up) = r + d*max Q'(s', a')
Q(lower left, up) = r + d*max (Q(upper left, right), Q(upper left, down))

Note that Q(upper left, right) > Q(upper left, down) because by moving right from the upper left state the agent gets closer to **G** then by moving down. This can be seen by

taking the most direct path to **G** from the state resulting from moving down from the upper left state and comparing its length to that of the most direct path to **G** from the state arrived when moving to the right from the upper left state. The max function dictates that the most direct path be taken.

Q(lower left, up) = 0 + .9 * Q(upper left, right)

Now the value for Q(upper left, right) needs to be found.

Q(upper left, right) = r + .9*max Q'(s', a')
Q(upper left, right) = 0 + .9*max (Q(upper middle, right), Q(upper middle, left), Q(upper middle, down))

Note that Q(upper middle, right) > Q(upper middle, left) and Q(upper middle, down) because it returns an immediate reward of 100. It is also the most direct route to **G**.

Q(upper middle, right) = 100

Now that the value of Q(upper middle, right) is known, the values can be substituted and the equation solved.

Q(upper middle, right) = 100
Q(upper left, right) = 0 + .9*100 = 90
Q(lower left, up) = 0 + .9*90 = 81

Figure 2.6 below shows the Q(s, a) values for each state and action in the six state world. It is important to note that in order for this diagram to be completed the agent would have had to try each path to the goal state, solving the Q function at each step. Looking at the diagram it can be noticed that the more direct the path is to the goal state the higher the reward.

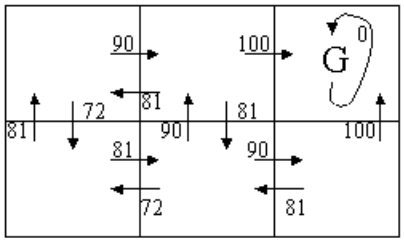

Figure 2.6. The Q(s, a) values for each sate in the 6 state world. For this example the discount value, d is set at 0.9. Notice that the highest Q values are returned for the most direct paths to the goal state.

Q Learning enables an agent to develop optimal strategies from delayed rewards even if the agent has no prior knowledge of the effects of its actions regarding its pursuit of its

goal. It does this by exploring the environment and solving the Q function at each step. The algorithm for Q Learning is given below.

Q learning algorithm

s - current state
s' - the state in the set of states adjacent to state s that contains a'. (s', a') is the optimal state/action combination adjacent to s.
a-current action
a' – action that returns the greatest reward at state s'
d-amount that rewards are discounted, range is a real number between 0 to 1
r – immediate reward available for current state.

For each state s, and each action, a, initialize the table entry Q'(s, a) to zero. Q' is used to indicate that since the algorithm has not visited all possible states, the Q values are expected values at this point. One point to note is that Q'(s, a) values will change (becoming more optimal) with time, as the environment is more thoroughly explored.

Observe the current state s

Do forever (until all states are visited):

- Select an action a and execute it
- Receive immediate reward r
- Observe new state s'
- Update the table entry for Q'(s, a) as follows:

$$Q'(s, a) = r + d * \mathbf{max}\ Q'(s', a')$$

- s = s'

Using this algorithm the Q table is built. The table for the diagram in figure 2.6 is shown below. Notice that the first entry has no action because it is the goal state. Once the agent is in the goal state it remains there, thus, no action is required.

Q table for six state world shown in figure 2.6.

Q(upper right, no action)	=	0
Q(upper middle, right)	=	100
Q(upper middle, down)	=	81
Q(upper middle, left)	=	81
Q(upper left, right)	=	90
Q(upper left, down)	=	72
Q(lower right, up)	=	100

Q(lower right, left)	=	81
Q(lower middle, right)	=	90
Q(lower middle, up)	=	90
Q(lower middle, left)	=	72
Q(lower left, right)	=	81
Q(lower left, up)	=	81

When the agent is operating it observes its state, s, and then locates in the Q table each Q(s', a) entry in which s = s'. It then selects the Q(s, a) entry with the greatest reward, Q(s', a'). By executing the action, a', the agent receives the greatest reward and gets closer to its goal state than if it had executed any other action associated with state s.

In summary, the main strength of Q Learning is that it allows a robot or agent to operate in its environment without a-priori knowledge. It does this by learning through the use of delayed reward. The results are stored in a table that maps state, action pairs to reward values. The entry in the table for state s that has the highest reward contains the action that moves the agent to its goal state quickest. The main weakness is that because the exploration of the environment is random many time steps are required before a complete Q table is formed.

2.4 Acoustically Guided Robots

This section provides a brief overview of work done using acoustically guided robots. Rule based methods of sound localization and investigation of the neurobiological processes in crickets and owls are discussed. The rule based methods[31] [32] were developed to see if robots could localize sound using two sensors, akin to human ears. The crickets[33] and owls[34] were studied for the specific purpose of determining the physiological structures and process used by these animals. While these works do not directly address methods of learning behaviors when presented with novel situations, they address important issues relevant to underwater formation maneuvering and learning. The rule based methods[31] [32] encountered many of the same acoustic issues encountered in the formation maneuvering task presented in this work including noise and directional ambiguities. The cricket phonotaxis work[33] used neural networks to control their robots, but the weights were solved for using a closed form solution instead of an adaptive technique. The robotic owl[34] work used a learning system to adapt to various situations, but it required many iterations in order to adapt to new situations.

Shah[31], and Kushleyev, and Vohra[32] developed a system loosely based on the human ability to localize a sound using two ears. Their system determines the direction of the sound source by examining both the amplitude and phase differences of the sound arriving simultaneously at two microphones. Their algorithm determines the direction of the sound by comparing amplitude and phase differences to a table of theoretically generated curves in order to determine the sound direction. They address the problems of range determination along with correct hemisphere detection (the process of determining

if the sound is in front of or behind the detection equipment) by using snapshots of data taken at separate time intervals. Using these techniques they developed a rule based controller that worked well in robot following tasks.

Reeve and Webb[33] used mobile robots to investigate cricket phonotaxis (the ability of female crickets to find male crickets by localizing on their song) in order to better understand the physiology and neurobiological processing in the cricket brain. They modeled the crickets using mobile robots with similar speed and hearing characteristics as crickets. To emulate the crickets neurological functions they used neural oscillators (a type of recurrent neural network whose key feature is excitory and inhibitory nodes) to process the sound and control the robots motion. The weights for the networks were found by solving for them using a closed form solution and then tuning the system for the lab conditions. Experiments were done incorporating the effects of various lighting conditions and oscillatory motions. Two different brain models were used, a less sophisticated, less physiologically faithful model, and a more complex, physiologically accurate model. In both cases the robots were able to seek to a sound source playing cricket songs. The more complex brain was able to more accurately emulate real cricket behavior.

Rucci, Wray, and Edelman[34] built a robotic barn owl in order to study the neurbiological structures responsible for localization behavior in owls of auditory and visual targets. Their machine consisted of a "head" with a centrally located camera and microphones on opposing sides. The head is mounted on an apparatus that accepts both elevation and azimuth commands. A neural network based control algorithm was designed to be neurologically faithful to the nervous system structures of a real owl. The neural network was trained using a reinforcement learning algorithm with the weights being updated using a modified version of Hebbian learning[35]. They report good results after training for a few hours during which about 15,000 targets are presented to the system.

2.5 Summary

This chapter presented background material for UUV teams and how they are related to robot formation maneuvering, robot architectures, learning systems used in robots, and acoustically guided robots. The next chapter documents the preliminary feasibility study undertaken to verify that FFNNs could be used to control autonomous robots in formation maneuvering tasks.

Chapter 3. Simulation of Multi-Robot Formation Maneuvering

In the previous chapter the importance of formation maneuvering as it relates to the UUV team mission was discussed. Realizing that the use of formations are key to the success of UUV teams, the goal of the work in this dissertation was to develop a machine learning approach that would endow robots/UUVs with the ability to create a formation and move to a destination in that formation. It was recognized that seeking and following leaders were two key abilities that would be needed in order to realize these goals, so this chapter discusses the pursuit of these sub-goals in the context of an adaptive system. The chapter starts by introducing the robot architecture for the system. Then, it continues by detailing the various aspects of the simulation such as the UUV model, the sensor model, the collision model, etc. Next, the formation maneuvering models are described, followed by a description of the learning system, and finally a short discussion of the results.

3.1 Conceptual Architecture

Fig. 3.1 below shows the conceptual diagram for a robot architecture whose key feature is the ability to generate behavior controllers on demand. This architecture served as a conceptual model for a system whose purpose was be able to continually pursue a pre-stated goal in a changing environment. This architecture is based on the three layer architecture described by Gat[23] and discussed in the previous chapter in that it has several functions active over varying spans of time. The next few paragraphs provide more detail on these functions.

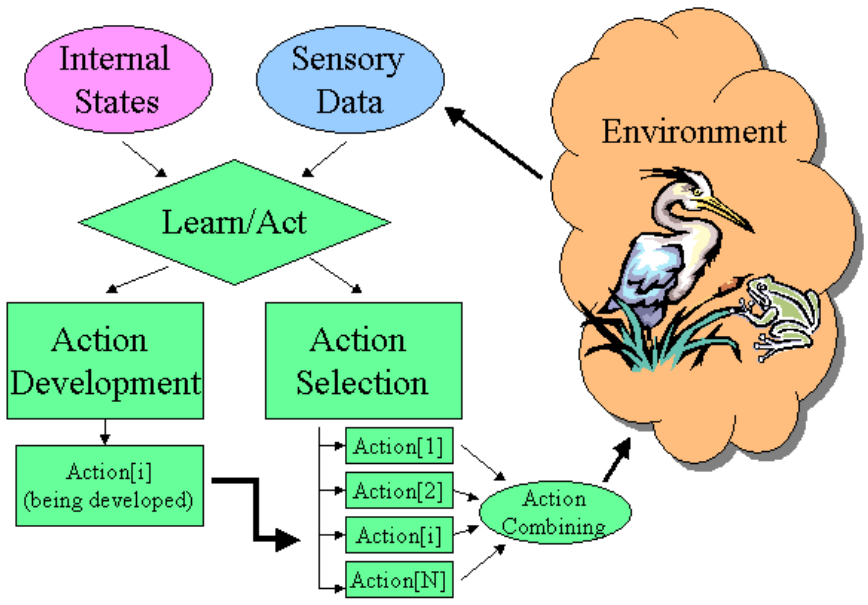

Fig. 3.1. A conceptual diagram of an adaptive system whose main feature is the ability to improve its controllers.

The actions are controllers that are low-level quick running processes that reside in the control layer. Example actions may include, obstacle avoidance, wander, seek to a sound source, or follow a sound source.

The Action Selection function fits into the sequencing layer because it controls which actions are currently enabled. The selection process could be either preprogrammed, as in Subsumption, or could be determined by an adaptive process.

The Action Development function is a long running module that best fits into the deliberation layer because it is primarily concerned with developing actions to be used in the future, i.e. when appropriate they will be selected by the Action Selection function. In this context, the method for developing the actions has to be a technique that can be reasonably executed by a machine process, and the range of the actions produced by the process must be flexible enough to be able to address the range of situations that the agent or robot will encounter.

Based on internal states and sensory data from the environment, the Learn/Act box decides whether the system needs to improve its controllers. Because it is also a long running process it is part of the deliberation layer.

Fig. 3.2 below shows the pared down system that served as the target robot architecture for formation creation and maintenance.

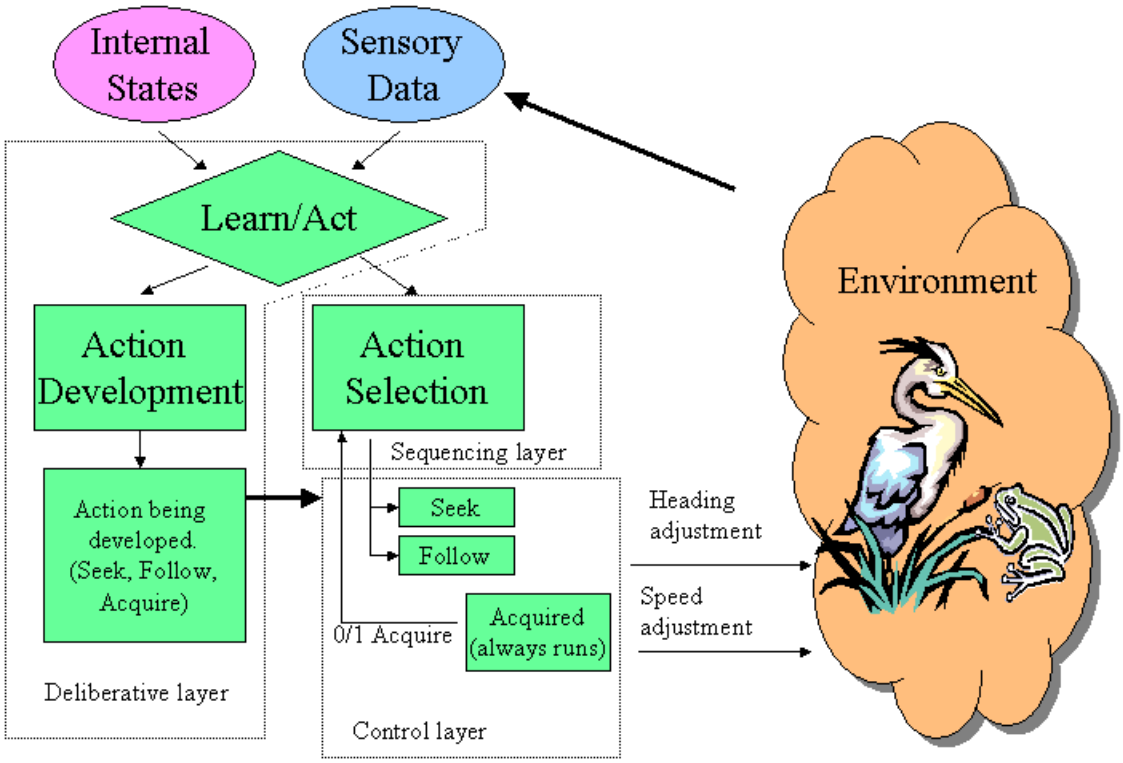

Fig. 3.2. Adaptation of the system in figure 1 was for the formation maneuvering problem. In this figure the different layers of the architecture are labeled.

The thrust of Fig. 3.2 is that by developing techniques to seek out a UUV and to follow a UUV various formations can be created and held. The effectiveness of the UUV can be increased if it can recognize which action is appropriate at the moment and have the ability to enhance the effectiveness of these actions for its current situation.

In regard to the type of controller that is to be used, a flexible, machine producible method of control is needed. Neural networks are used as controllers because it is a more straightforward process to develop a program that produces usable neural networks than to create programs that write programs using symbolic methods (coding of algorithms in a language such as C or LISP). While it is possible to write programs that write other programs, the process involves making sure that the machine produced programs are syntactically correct and that they run in an environment in which they can be contained. Feed forward neural networks were selected because these problems can be avoided and their hidden layers afford them more flexibility than other neural networks[36] such as ADELINES or Perceptrons. This increased flexibility gives them the ability to represent boolean functions, continuous functions, and arbitrary functions.

For this stage of the research a bottom up approach was used and it was decided that it would be prudent to develop the key parts of the system in order to make sure that they would suffice for the task at hand, formation creation and maintenance. Specifically, there was a need to show that feed forward neural networks would suffice as controllers in formation maneuvering tasks. In order to realize this goal, a multi-robot simulation system was developed in order to test the controllers in formation maneuvering tasks. The architecture presented earlier is a design tool used to identify the modules that are needed in the overall system, but has not been fully implemented. The next section describes the simulator that was developed for testing of the neural network controllers.

3.2 Multi-Robot Simulator

A multi-robot simulator was a research tool developed by the author for this stage of the work. This simulator uses a variant of the Genetic Algorithm (GA) to grow feed forward neural network controllers for simulated UUVs in a simulated environment. It uses LabView as a control and display engine and C as the calculation engine. The relevant input parameters for the controllers, the lead UUV and the display of the simulated environment are managed by LabView, while the calculations of the various sensor values, environmental effects, and control algorithms are implemented in a library written in C and accessed by LabView.

Fig. 3.3 below shows the main control screen of the multi-robot simulation program. The program operates in two modes, training of controllers, and testing of the controllers using formations. The graph in the top part of the figure shows the error incurred while training controllers. In test mode, the two blue level meters to the lower left of the figure indicate the signal strength of the left and right sensors of the robot piloted by the operator while the dial to the lower right of the figure is the directional control of the lead robot, piloted by the operator.

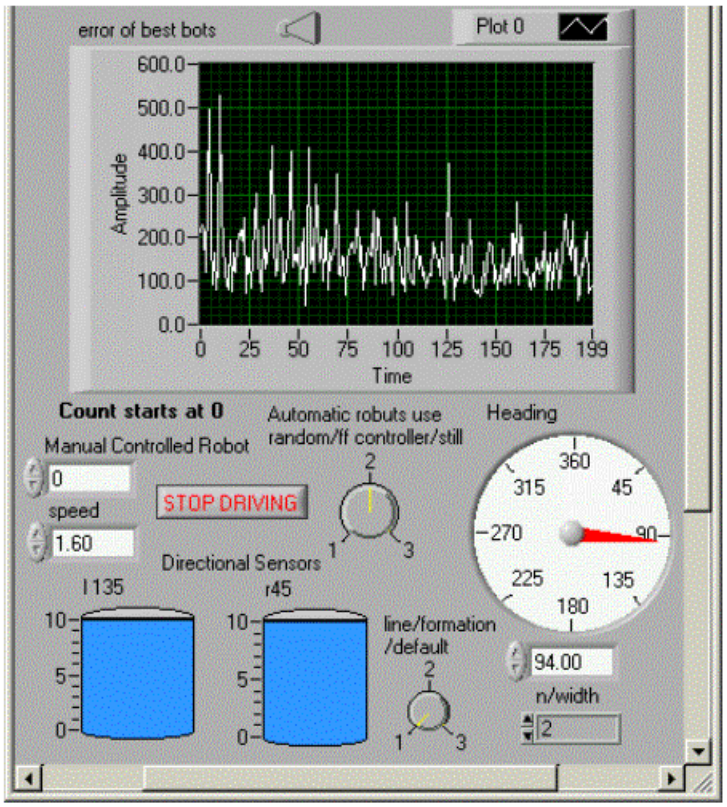

Fig. 3.3. Part of the main control screen for the formation maneuvering simulator. The graph shows the error level of controllers during training mode. The y-axis of the error level graph is the distance between the follower robot and the lead robot. The x-axis is the generation number. This particular run shows that the error has gradually decreased as the number of training generations increased. The blue meters on the lower left of the screen show left and right acoustic sensor levels of the manually controlled lead robot. The large dial to the right is the directional control for the manually controlled robot. These meters and the directional control dial are used in formation testing mode, not training mode.

Key features of the multi-robot simulator are:

1. UUV model. The robots are loosely modeled after the Underwater Navigation Control Lab (UNCL) lab robots (ActiveMedia Pioneer 2DX) in that they have the same speed characteristics, top speed of 1.8 m/sec is enforced, but in the interest of simplicity, acceleration characteristics were not modeled. The lab robots can turn in place so there are no turning restrictions applied.
2. Sensor model. See figure 3.4 below.
    a. Two omni directional sensors placed at 45 and 135 degrees on the robot at a range adjustable from about 0.25 to 5 meters from the center of the robot. The range is adjustable so that amplitude differences between the microphones can be easily accentuated or dampened, much like a gain control. The front of the robot is at 90 degrees.

b. Each robot has an ambient amount of noise and noise associated with the speed that it is moving. The faster it moves, the noisier it is. That is, there is a minimum amount of noise that a robot makes even when it is not moving. This feature allows a follower robot to seek to a non-moving robot. By increasing the volume as the robot moves, the follower robot can still track a robot even if the distance between the two is growing due to speed of the leader.

c. The signal strength is modeled using a spreading model which is based on introducing a finite amount of energy into a system and calculating its density across a uniform area. Signal strength decreases with the square of the distance[37] between the source and the receiver. The equation for the strength at the sensors in shown in figure 3.4 below.

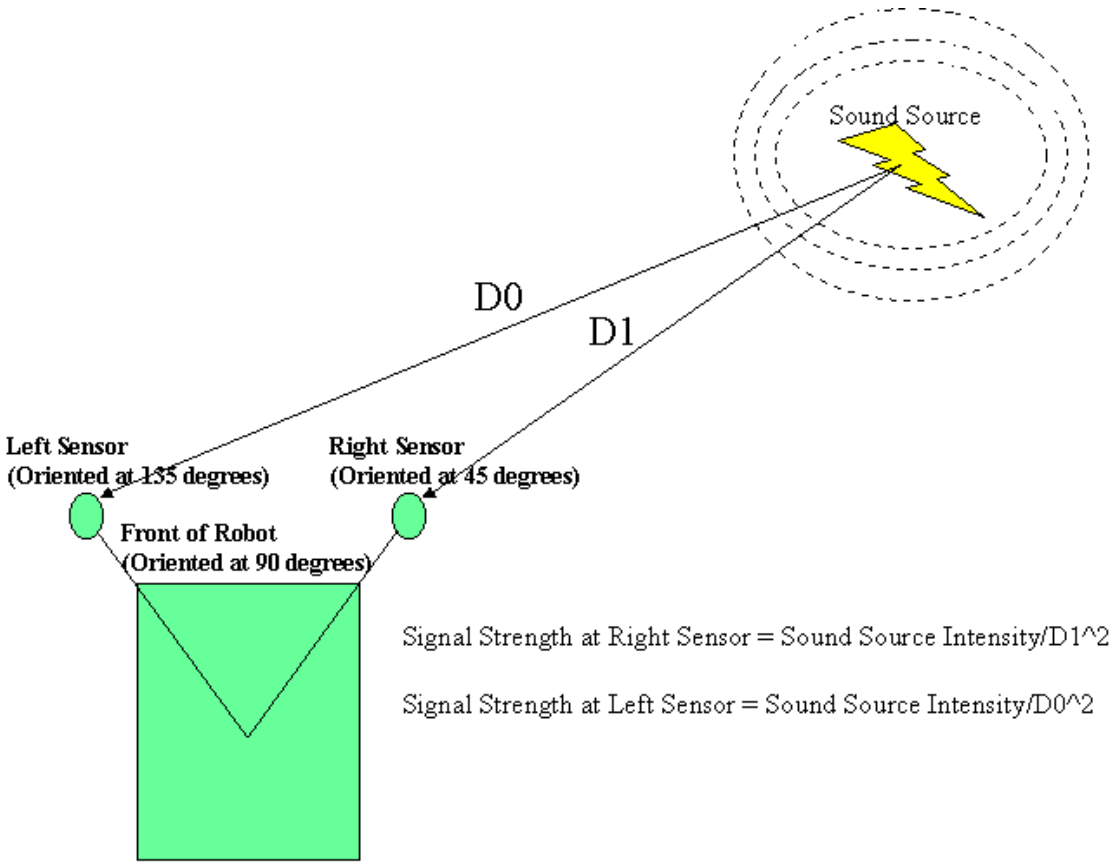

Fig. 3.4 The robot with its two sensors at a 90 degree angle from each other. Because the distance from the source to the right and left sensors is different, there will be a difference in the strength of the received signal at the two sensors.

3. Communication model. Each robot has a signature frequency that can be used to identify the robot. This model is commonly referred to as a frequency multiplexed model. In this system each robot is assigned a portion of the total usable frequency band. For example, robot 1 may use 300Hz to 500Hz, robot 2 may use 700Hz to 900Hz, and so on. In this system the follower robot does not actively guard against miss-identification of its leader; if another robot or object is

making noise in the followers assigned frequency range, error can occur. Because the purpose of this system is to test the effectiveness of the neural network this issue was not addressed. Since this is a simulation, the actual frequencies do not have to be assigned. Instead, it is enough to calculate the signal strength of the energy emitted by a leader robot received by the sensors of the robot assigned to follow it.

4. Collision model. When robots collide they react according to a physical model with the following characteristics.
    a. Inelastic model collisions enforced. No energy is absorbed due to deformations of robots. This model is likened to billiards.
    b. Robots are modeled as round point masses.
    c. Collision radius is set at 1 meter. Although the real robots are dimensionally smaller, the 1 meter model works well for visualization purposes.

This simulation was used to show the viability of neural networks as controllers in formation maneuvering task. It does not accurately model environmental conditions that are seen in the lab, such as sensor/source directivity, harmonics generated by equipment, and reflections and multi-path of sound.

The next section describes how the simulated robots were used to create and maintain line and tree formations. It shows conceptual diagrams of the formations and finishes with some screen shots of the simulation.

3.3 Leader/Follower Approach Using Relative Navigation

Line and tree formations were developed using relative navigation and the leader/follower approach. Leader assignment for the line formation is as follows. Each robot follows the robot with id one less than its own id. Fig. 3.5 below shows the geometry of a robot line using this scheme.

The lead robot, the green smiley face with Id = 0 shown in the lower middle part of figure 3.5, is manually piloted by an operator. Its relative location is sensed by the robot whose id is 1. The robot whose id is 1 never knows exactly where either the green robot is or for that matter what its own position is. It only perceives its relative position in relation to the green robot, although with the approach used it never actually computes its relative position. Its controller uses the sensor inputs to minimize the distance between the two robots. In general, if the lead robot is to one side or another of the follower, the follower turns towards the leader. If not, the follower heads straight. This method, as discussed in chapter 2, is an implementation of the leader/follower concept. In this implementation, it uses passive relative navigation. It is relative because there are no absolute coordinates being transmitted from the leader to the follower, and passive because the follower senses the signal that the leader makes and heads in that direction, instead of receiving instructions, such as head left, or head right. This method was chosen for its simplicity and robustness. At this point in the research, it was assumed that the underwater

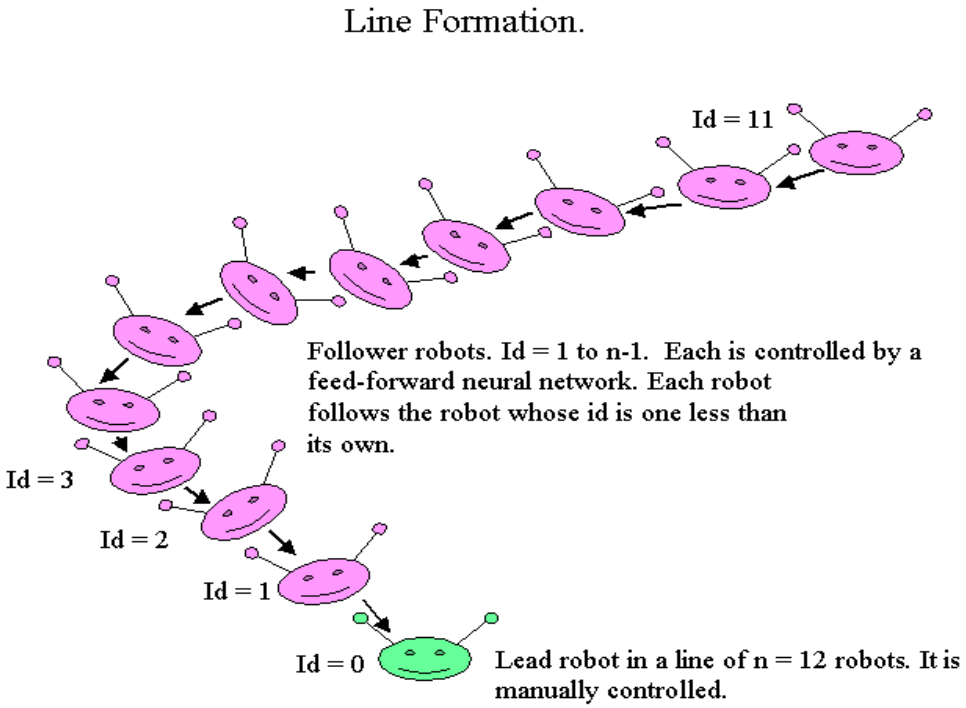

Fig. 3.5. Robot line containing 1 lead robot and 11 followers. The green robot is controlled manually from the robot school control screen. Using their sensors and controllers, each robot follows its assigned leader.

environment would cause complications for any encoded information transmissions, so the aforementioned method was designed to be a simple, robust, and low information content methodology. The line formation is simplest because each robot follows only its assigned leader. The formation remains stable as long as no robot loses its leader and the formation leader is operating correctly. In this strategy, if any robot loses its leader, or the formation leader breaks, all of the robots in the formation will not reach their goal. Other researchers have addressed these problems[12]; they are not addressed here because the goal was to establish that neural networks would suffice as controllers in a formation maneuvering task using sensors similar to those that would be available in an undersea environment.

In tree formations, leader assignment is similar. Each robot follows the robot that is at the root of its subtree. Fig. 3.6 below illustrates the leader/follower relationship of the tree formation.

Stable tree formations were created by having each robot being attracted to (following) the robot with its own id divided by two, and at the same time being repelled from any robot with an id within 1 of its own. Looking at figure 3.6 it can be seen that robots on the same level of the tree all have an id within one of each other. Said another way, robots follow the parent nodes in the tree and avoid the leaf nodes on their own level.

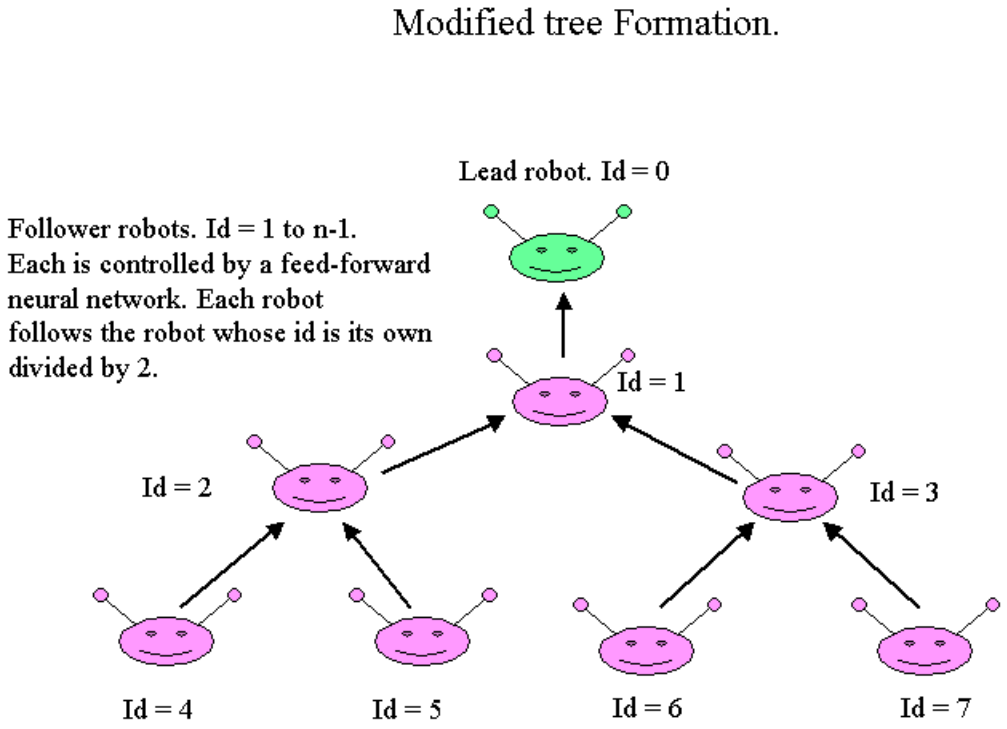

Fig. 3.6. The leader/follower relationship in the tree formation. Note that follower robots identify their leader robot by dividing their own id by 2 using integer division. For example when robot 4 needs to know its leader, it divides its id, 4, by 2 (4/2 = 2) resulting in 2, so robot 2 is its leader. For robot 5, the process is similar, 5/2 = 2.5, but in integer division the fraction is dropped so once again the leader is robot 2.

The next section describes the feed forward neural network controller structure and the genetic algorithm used to find the relevant parameters in the controllers that the simulated robots use.

3.4 Controller Structure and Learning Algorithm

The feed forward network[38] [39] used by the robots in the simulator has 2 nodes in the input layer, 4 nodes in the hidden layer, and 2 nodes in the output layer. It is worth noting that hidden layers of up to 12 nodes were tested but no performance gain was seen. Tesauro[61] used up to 40 nodes in the hidden layer in his work and he reports that after many (over 1 million) training iterations the extra nodes helped in the learning subtle nuances. This was not investigated in this work because the goal of this work was to produce a draft controller on the fly, with emphasis on producing the controller in time to be used in the current situation.

The inputs are from the left and right acoustic sensors. The outputs are heading and speed adjustments. Both the inputs and outputs are mapped to stable ranges. That is, there are minimum and maximum input values. If the values go beyond these ranges, they are truncated so that they fall within the acceptable range. A similar situation exists

with the output values of the network, they are held within a pre-defined range. The network itself is fully connected. Figure 3.7 below shows a diagram of the network architecture.

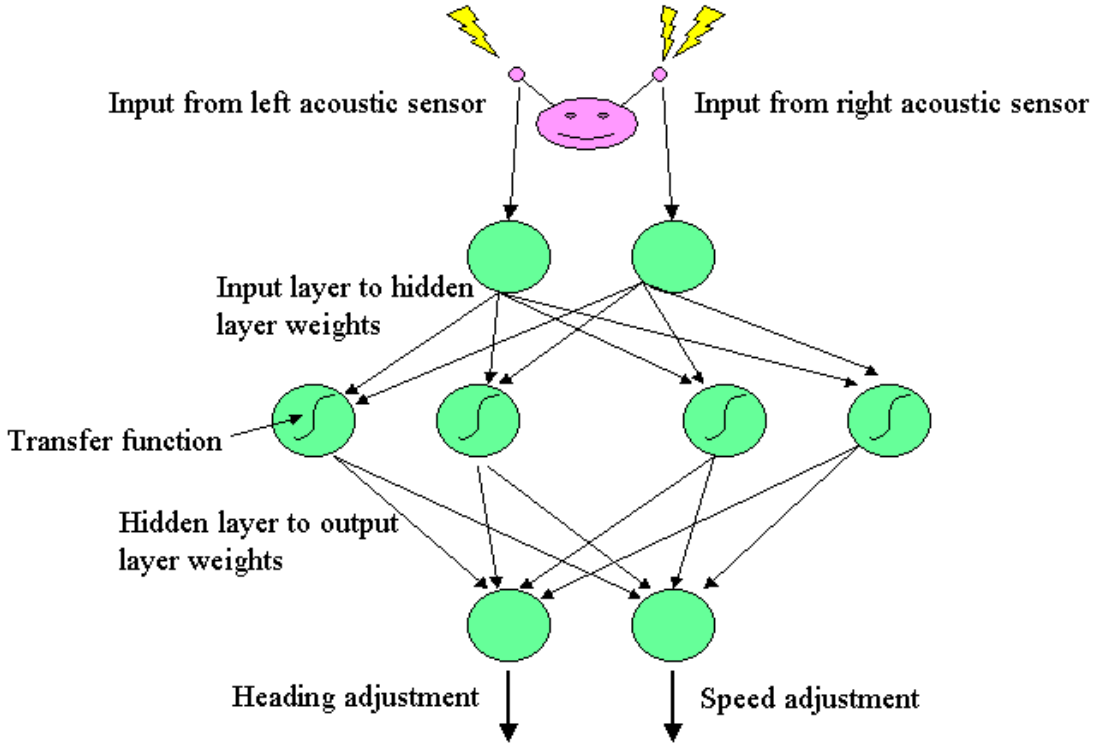

Fig. 3.7. The geometry of the feed forward network used in both seek and follow modes. It has 2 nodes in the input layer, 4 in the hidden layer, and 2 in the output layer. The weights and transfer functions are selected by the GA.

In this work the GA was used to find the weights of the neural networks. It was selected over more traditional methods such as Back Propagation because the work in this dissertation seeks a general learning method that is not bound by needing explicit desired values for each training example. In the current discussion of formation maneuvering, this data could be acquired from the simulator, but for other applications, such as gait development in walking robots, the correct motions at each moment are often not known. In these instances a GA can use a fitness function that promotes overall system performance such as walking without falling, even though it does not have step-by-step knowledge.

Because it is not always easy to determine which transfer function would work best for a given problem, in this research the genetic algorithm also selects the type of transfer function that will be applied at each node of the hidden layer. It can select from linear (no transfer), discrete (if the input is greater than 0 then output is 1, otherwise it is –1), and sigmoid ($y = 1/(1+\exp(-v))$). So in this implementation, transfer functions in the hidden layer can vary within a network.

While including network transfer functions as part of the optimization algorithm is not widely covered in texts or the literature, there is some indication that it has been experimented with. For example Zhao[40] reported using a GA to optimize the number and placement of radial basis nodes in their modular neural network. That work is different than this work in that the GA worked with node placement and topology, while the weights were found with more conventional back propagation techniques.

In his overview of work on Particle Swarm Optimizations (PSO) Hu[41] indicates that evolutionary computation methodologies have been applied to finding transfer functions in neural networks, but does not list specific instances.

For the research in this dissertation using the GA to solve for transfer functions is vital. Initial tests using sigmoid functions and linear functions experienced convergence problems. With no other changes, application of Perceptron[42] type discrete transfer functions resulted in a marked, and surprising improvement. This realization that it would be easy to make an incorrect transfer function selection in an otherwise correct arrangement of a neural network motivated use of GA selected transfer functions.

From an implementation standpoint finding the transfer functions with the GA is not difficult. To do so requires one extra chromosome that holds enumerated values that indicate which transfer function to use. The chromosome is initialized randomly from the set of enumerated values, and is treated as all other chromosomes are during the breeding process. In a typical network there would be a mix of transfer functions, usually mostly discrete, with one linear or sigmoid in the mix.

For the work in this dissertation, the variation of the GA used is similar to the Elite GA described in Brooks and Iyengar[43]. The main difference between the Elite GA and the standard GA is that the Elite GA creates the next generation exclusively from the best scoring portion of the current population. Using the 20 percent as an example, if the population is 100 in size, an Elite GA would transfer the 20 best individuals of the population to the next generation and create the remaining 80 by breeding the top 20 individuals. In this scheme, the lower portion of the population gets no chance to contribute to the next generation. In a standard GA, all individuals get a chance to contribute, proportional to their fitness. A brief outline of the Elite GA algorithm that is used to find the weights and transfer functions of the feed forward neural network controller used for following is given below:

The variables are defined as follows:

- n : population size

- P : a population of size n of feed forward neural networks;
  P = {p(0), p(1), …..p(n-1)}

- Let each p(i) in P be an individual that consist of a weights chromosome w, and a transfer function chromosome t. p(i) = {w, t}.

- Each w in p(i) contains the following number of weights:
  - (number of inputs * number of hidden nodes) + (number of hidden nodes * number of outputs)
  - The range is floating point –1.0 to 1.0

- Each t in p(i) contains the following number of enumerated elements :
  - number of hidden nodes
    - The range is integer 0 to 2
    - Let an element of t = 0 to indicate the discrete transfer function.
    - Let an element of t = 1 to indicate the linear transfer function
    - Let an element of t = 2 to indicate the sigmoid transfer function.

- nsteps : number of time steps over which the fitness function evaluates an individual. Typically in this work the range is from 100 to 300 time steps.

- Let Fitness(p(i)) be a function that measures the fitness of the individual, i, who is a member of population, P. The fitness function measures the cumulative distance between the follower robot and a lead robot over the specified number of steps, nsteps. The follower robot is controlled by the neural network described by individual p(i), while the simulation system controls the lead robot.
- scores : array to keep track of the fitness results of the population members.

- M : maximum number of generations to be executed.

- P0 : current population.
- P1 : next population.

- topPercent : percent of the P0 that gets transferred to P1. It is a user adjustable parameter with a typical range of 10 to 33 percent.

The algorithm is shown below:

```
/* Create an initial population, P0. */
For (each p(i) in P0) {
        Randomly initialize each w(i) in w
        Randomly initialize each t(i) in t
}

/* Run the simulation for M generations. */
For (j=1 to M) {
        /* Evaluate the population using the simulation in the Fitness function. */
         For (each p(i) in P0) {
                scores(i) = Fitness(p(i))
```

```
        }

        /* Using the scores provided by the Fitness function, rank the population from
        best to worse using a sort.  The best individual is in position 0 and the worst in
        position n-1 of the scores array. */

        sort(scroses)

        /* Create a new population P1 from the most fit individuals of current population.
        */
        Transfer the topPercent portion of P0 to P1.

        Use the topPercent portion of the P1 to breed individuals for the remaining
        portion of the new population P1.
                • Using crossover and mutation functions transfer the genetic
                  material in chromosomes w(i), t(i) from the selected parents to the
                  child.
        / * Replace P0 with P1.  */
        P0 ← P1;
} /* End for M. */
```

Return the individual from P0 with the highest fitness value.

The benefits of using this method are its stable operation and straightforward implementation[43].

In a typical simulation session, the user will use this algorithm to grow a controller. Usually the GA will build a working controller within 200 generations. The low number generations is important in the sense that the controller can be generated in time to be useful in the current situation. After the controller has been grown, it is tested by using a 2 UUV simulation, and stored for later use (see Appendix A for an example). During a testing session, the user will control the lead UUV and the just grown controller will control the follower robot/robots. Once the user is satisfied that just grown neural network controller is working, i.e. the manually controlled UUV will be followed by the computer controlled UUV, a multi –robot formation can be tested using the controller. Recall that this part of the work was done only to prove the viability of the neural network as a usable controller that could be generated by an automated process, not as a preliminary design for a fieldable system. Currently, both line and tree formations have been developed, with line formations being easier to create and maintain than tree formations. Appendix A shows an example of a saved controller for a following robot.

The next section provides results acquired using the multi-robot formation maneuvering simulator. Data from the controller creation phase is presented along with screen shots of selected formations. Inter-robot distances during formation creation and maneuvering are also presented.

3.5 Experimental Results

This section is divided into two parts. The first part discusses the parameter settings used in a typical training session and continues by showing fitness function results as a function of generation. The next part shows results obtained during formation creation and maneuvering. In this section inter-robot distance is discussed along with the presentation of several screen shots taken during maneuvering sessions.

As stated in the previous section, a typical training session was 200 generations long. Typically a population size of 30 was used along with a mutation rate of 30%. Other researchers[44] report typical population sizes of 100 to 1000 individuals and a mutation rate of 0.01%. The mutation rate is set to this high level because the original software was developed on less powerful computers. In these experiments, by limiting the population size, and turning up the mutation rate, it was possible to build adequate controllers more quickly than using a larger population and a smaller mutation rate.

As mentioned earlier, during the training session the lead robot is controlled by the computer. Initially the lead robot and the follower robot are placed 1 meter apart. When the simulation starts, the computer steers the lead robot, making random speed and direction changes, resulting in a different test being made for each robot. The test continues for a predefined number of steps, usually between 180 and 360. At the end of the test, the average distance between the leader follower pair is recorded.

By using a random path specialization by the networks is avoided, that is, since all the tests are different, a network that can only learn one function, such as turn left cannot be created. The disadvantage of this technique is that it is hard to numerically compare networks. If a network scores just slightly better than another, it could be that it is a better network, but it also could be that its computer controlled leader was harder to follow. To compensate for this, many extra generations are run, helping to ensure that the controllers are stable. In the graph shown in Figure 3.8 below, it can be seen that after about 50 generations there is no measurable improvement in the effectiveness of the best controllers from each successive generation. In the earlier generations, from 0 to 30, it can be seen that the distance between the computer controlled leader and the best follower in the population narrows from about 1.1 meters to about 0.5 meters.

The next paragraphs discuss the formation maneuvering tests conducted using the generated controllers. In these tests, typical robot line sizes ranged from 5 to 20. Tree formations ranged from 3 robots to 8. Data showing inter-robot distance is presented along with several screen shots of various formations.

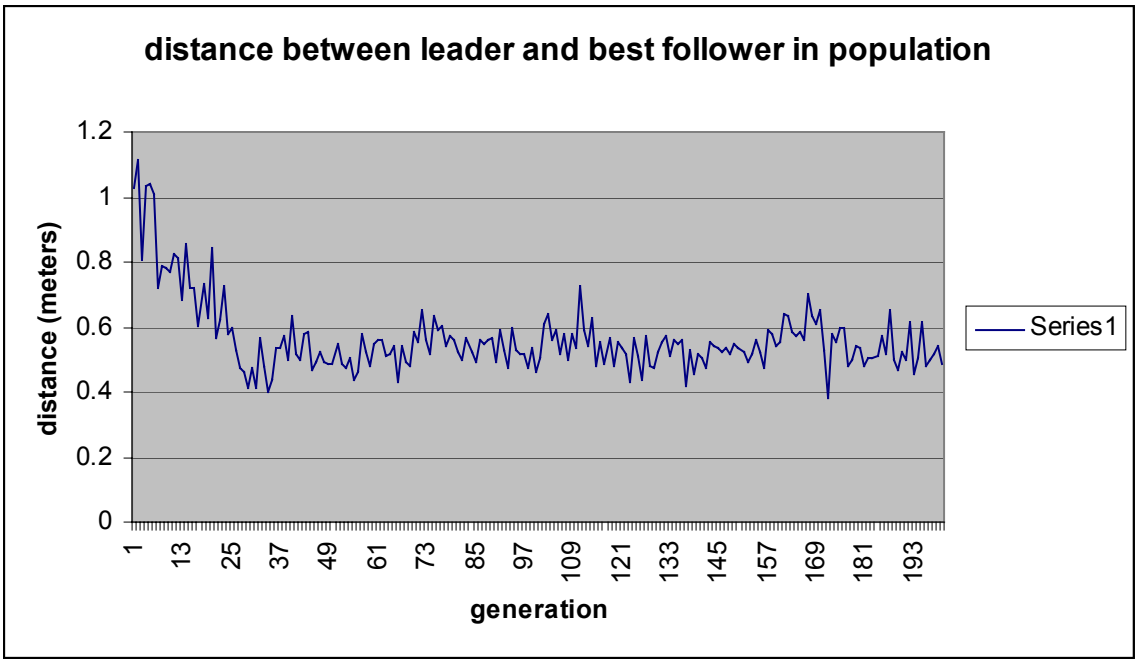

Figure 3.8. Performance in following distance versus controller generation. The y axis is the average distance that the feed forward neural network controlled robot (best controller from that population) maintains behind a computer controlled lead robot that is randomly changing its course and speed. The x axis is the generation number. Fluctuations in the curve are due to each fitness test being different because of the random course changes. This plot was created by averaging the results of 10 training sessions at each generation.

The effectiveness of the controllers in line formations were tested using a simulator setting in which the operator piloted the lead robot, and a neural network controller piloted each of follower robots in the line. The session starts when the operator pushes the "Drive Robots" control. At this point the robots are put in an initial position, a horizontal line in which they placed one meter apart and given random directions and speeds. In order for the line formation to successfully be created they must orient themselves so they are facing the back of their leaders, before their leaders move to a range in which the amplitude difference at the sensors is too small for the controller to discern.

Figure 3.9 below shows a plot of the distances between the robots of a line formation. To create this plot a 10 robot formation was maneuvered in an arc across the screen. The plot shows the inter-robot distance between the first 5 robots. The x-axis is time samples. Each time sample is 20 milliseconds long. The y-axis shows the distance between the robots. The dark blue curve labeled dist01 is the distance between the formation leader and the 1st robot. The magenta curve labeled dist12 is the distance between the 2nd robot in the formation and the 3rd robot, and so on. As can be seen from the plot, in the early stages of the formation the robots are close to one meter apart. At this time the follower robots are orienting themselves to their leaders. Afterwards the distance opens up and becomes stable as the formation begins to maneuver away from its starting point.

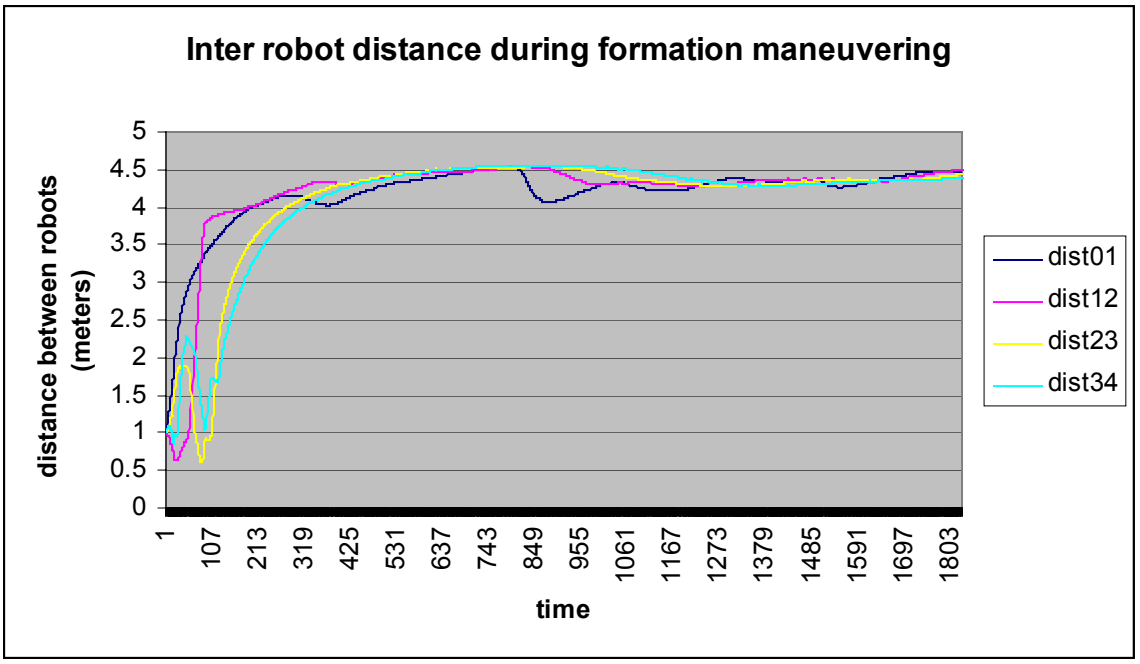

Figure 3.9. The inter-robot distance during a formation maneuvering session with a line formation of 10 robots. The distances between the first 5 robots are shown. The dark blue curve labeled dist01 is the distance between the formation leading robot and the first follower. The magenta curve labeled dist12 is the distance between the first follower and the second follower, and so forth. The lead robot is moving with a velocity of 1.8 m/s.

Again looking at the plot, the orientation phase is from about time slice 1 to time slice 150. From time slice 150 to about time slice 650 the line is stretching out. From time slice 650 onwards the distances remain relatively stable. At about time slice 950 there is a fluctuation in the distance between the lead robot and the first follower, but it is not readily apparent in the other curves. This fluctuation was caused by an abrupt movement of the directional control of the lead robot by the operator. The fluctuation is less noticeable in each successive curve, i.e. it can be seen in the inter-robot distance curve for robots 1 and 2, but it is less pronounced and it occurs at a slightly later time, and so on. This phenomena is similar to what happens in automobile traffic when someone makes a sudden stop. The person directly behind them has to react the quickest to avoid an accident but with each successive car in the line the reaction time required is lengthened. The formation from which the data in figure 3.9 was collected is shown in figure 3.10 below.

The majority of the formation maneuvering work done in simulation used controllers developed for following. Seek controllers were tested, but not in formations. Because the full robot system architecture was not implemented it was not possible to switch from one action to another. In testing with one leader and one follower, the behavioral difference between the two networks is that the follow network can keep the following robot in very close proximity to the leader, whereas the seek network tends to wander more and can loose track of the leader much easier. The wandering causes the robot to cover more ground in its search for the leader, making it more suited to situations in

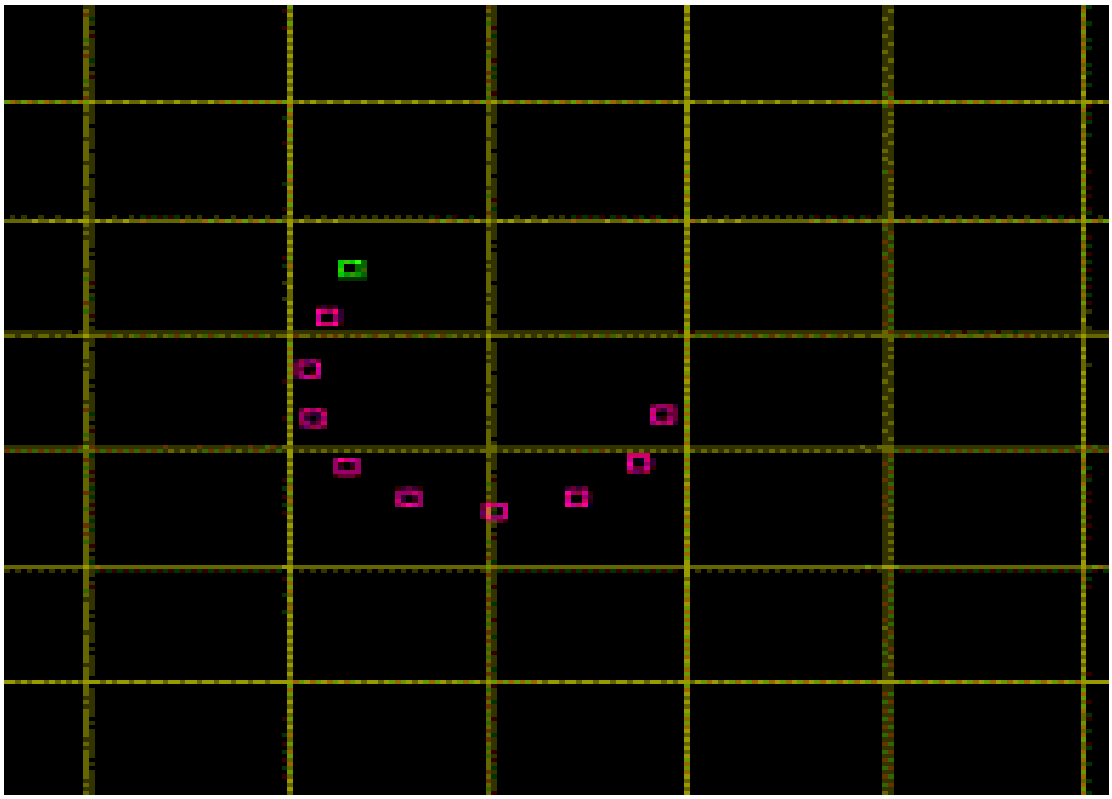

Figure 3.10. A line formation of 10 robots. The plot shown in figure 3.9 was derived from the distances between the first 5 robots in this formation. This picture corresponds to the last time slice in figure 3.9. The green square is the lead robot. The grid lines are at 10 meter intervals.

which the lead robot is not close to the follower. When a follow network looses track of its leader, it becomes stationary. Using the seek controller, the robot will keep moving, whether the leader is close or not.

These behaviors are a direct result of the fitness functions. The fitness rule for the follow function rewards proximity to the leader, whereas the fitness rule for the seek function rewards both proximity to the leader and area covered during the search for the leader.

In a line formation, the formation is stable as long as no robot loses its leader. Because there is only a single direction of communication between leaders and followers, leaders do not know if followers are locked on to them. While implementing a system of acknowledges into the inter robot communication strategy would help alleviate this problem, the goal of this work was to establish that neural network controllers that would suffice for formation maneuvering tasks. The virtual structure approach developed by Tan and Lewis[12] has been shown to be robust in the presence of failing robots, but it is rule based, making it more difficult to adapt to environmental changes, and it relies on an external system to provide vessel positioning, which would not be suited for undersea work.

Using the simple single direction communication based follow controllers, lines of 20 robots have been maneuvered for up to 20 minutes in the simulator. Lines of up to 30 robots have been tested, but because of reasons detailed above, it is difficult to establish the formation. Fig. 3.11 below shows 20 robots forming a line; the green robot is the manually controlled leader. This figure shows the line soon after the robots have moved out of the initialization phase. This figure corresponds to about time slice 150 on figure 3.9.

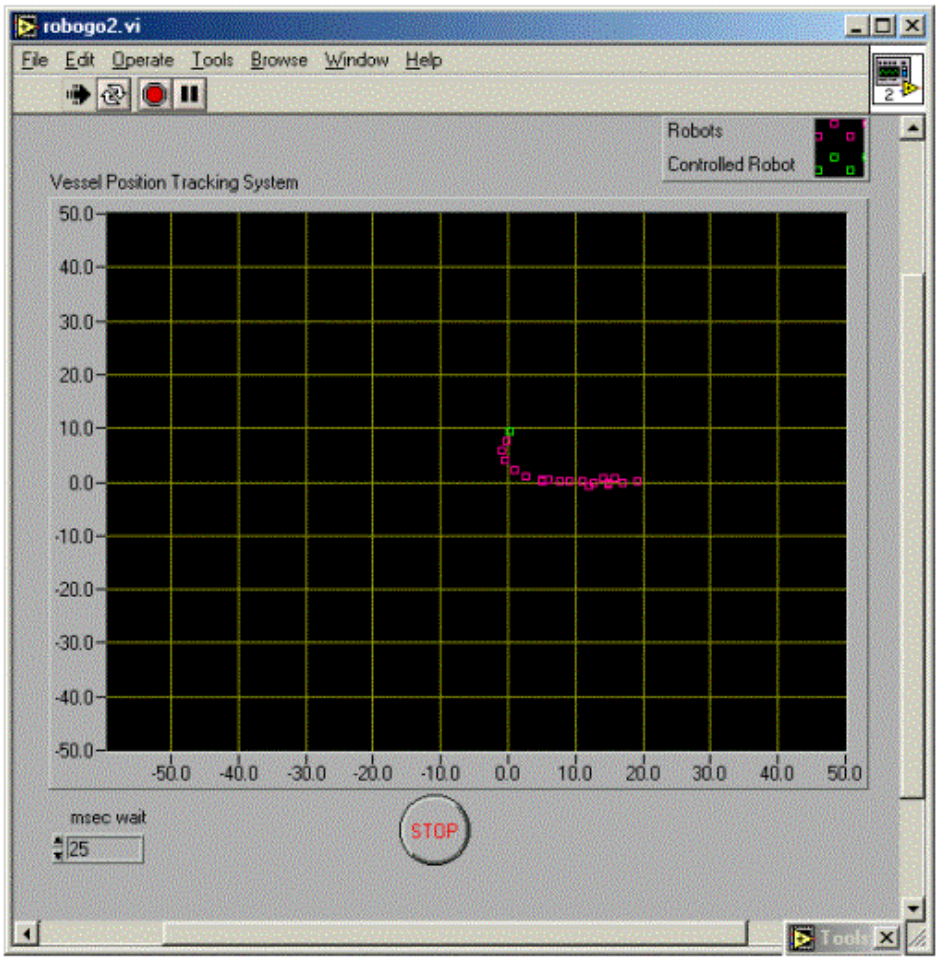

Fig. 3.11. A line of 20 robots attempting to form a line. The green square is the leader robot.

As can be seen in figure 3.11 the robots have not moved too far from their initial positions along the X-axis, as they are in fairly close proximity to each other. At this moment the robots are milling around trying to get their bearings on their assigned leaders. Several collisions occur because obstacle avoidance has not been worked into the fitness function, or architecture. In some cases the collisions may help in the line formation, because they help keep the robots bunched together long enough for each robot to find its leader. As the lead robot is manually driven away from the formation creation area, the robot line emerges. Figures 3.12, 3.13, and 3.14 show various lines of robots being maneuvered on the screen.

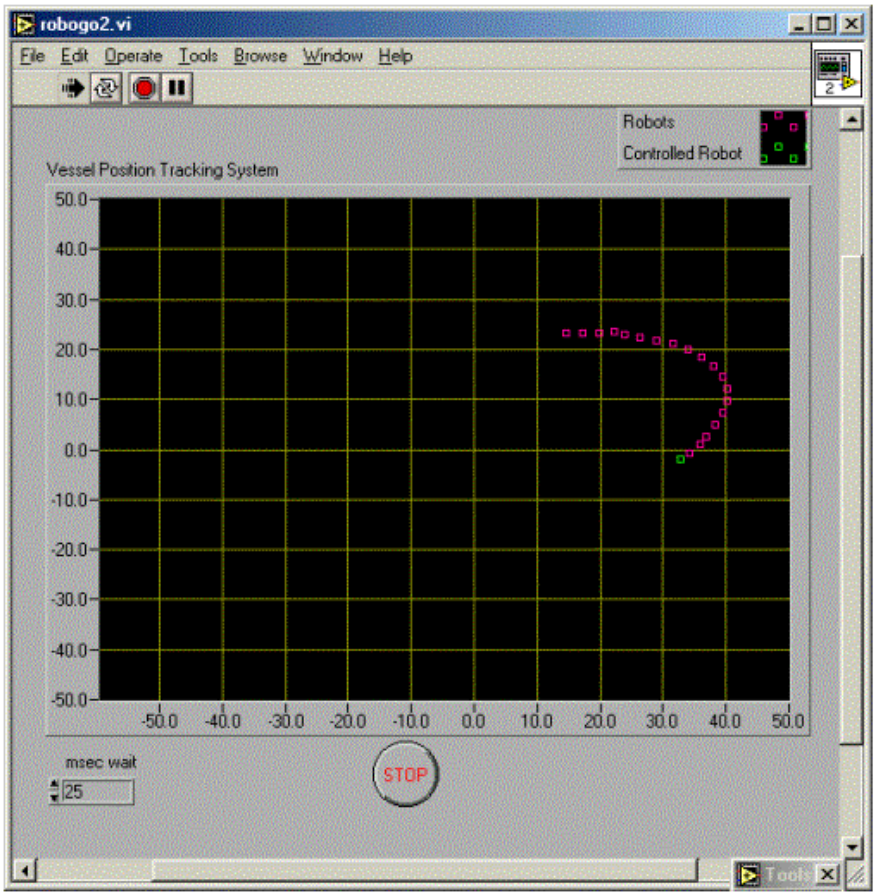

Fig. 3.12. Line of 20 simulated robots/UUVs making a sweeping turn.

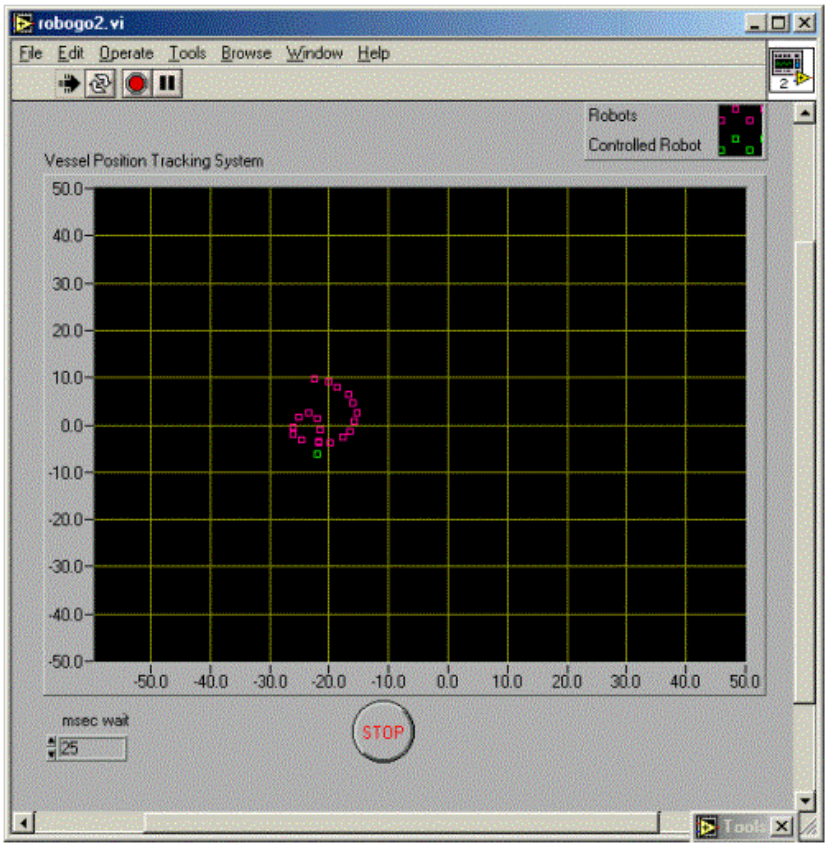

Fig. 3.13. A loop in a line of 20 robots. As the line crosses itself the robots usually collide and bounce off each other. Usually they can remain "locked on to" their leaders keeping the line in tact.

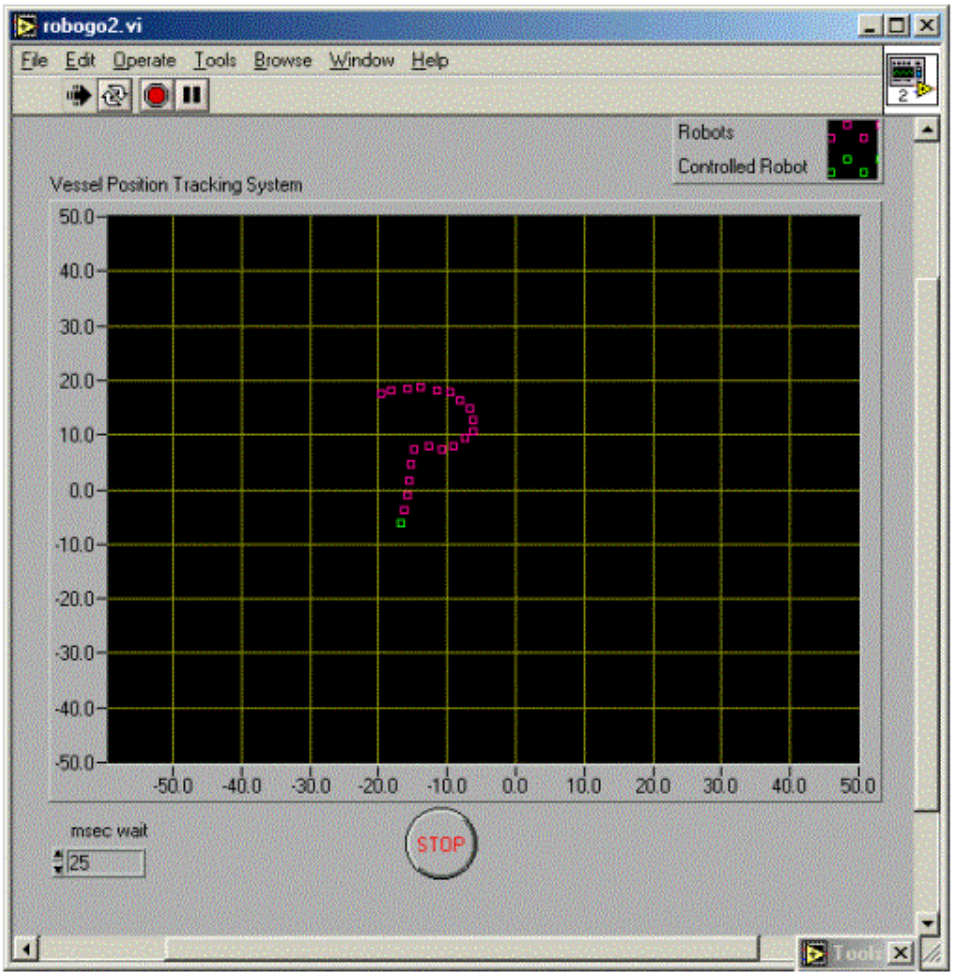

Fig. 3.14. A formation that looks like a question mark. The sharp turn was the result of a collision that occurred when attempting to make a loop in the line.

As stated earlier, tree formations are more difficult to form and maintain. Figure 3.15 below shows a tree formation of 8 robots heading across the screen from right to left. Notice that the last 4 robots in the formation are very close to one another, a potential formation destabilizing problem. The first level with more than one robot is stable because the robot 2, the one on the lower side of the screen in level with two robots, is attracted to robot 1 and repulsed by robot 3, the other robot on that level. Robot 3 is in a similar situation, it is attracted to robot 1, and repulsed by robot 2. The next level, the level with 4 robots tightly grouped, is similar. The bottom robot, number 4 is attracted to robot 2, and repulsed by robot 5. Robot 5 is also attracted to robot 2, and repulsed by robot 4. Here in lies the problem, the software used in this diagram did not cause robot 5 to be repulsed by robot 6, or robot 6 by robot 5, allowing the sub-trees anchored by robots 2 and 3 (see diagram in figure 3.6, come close to each other or collide.

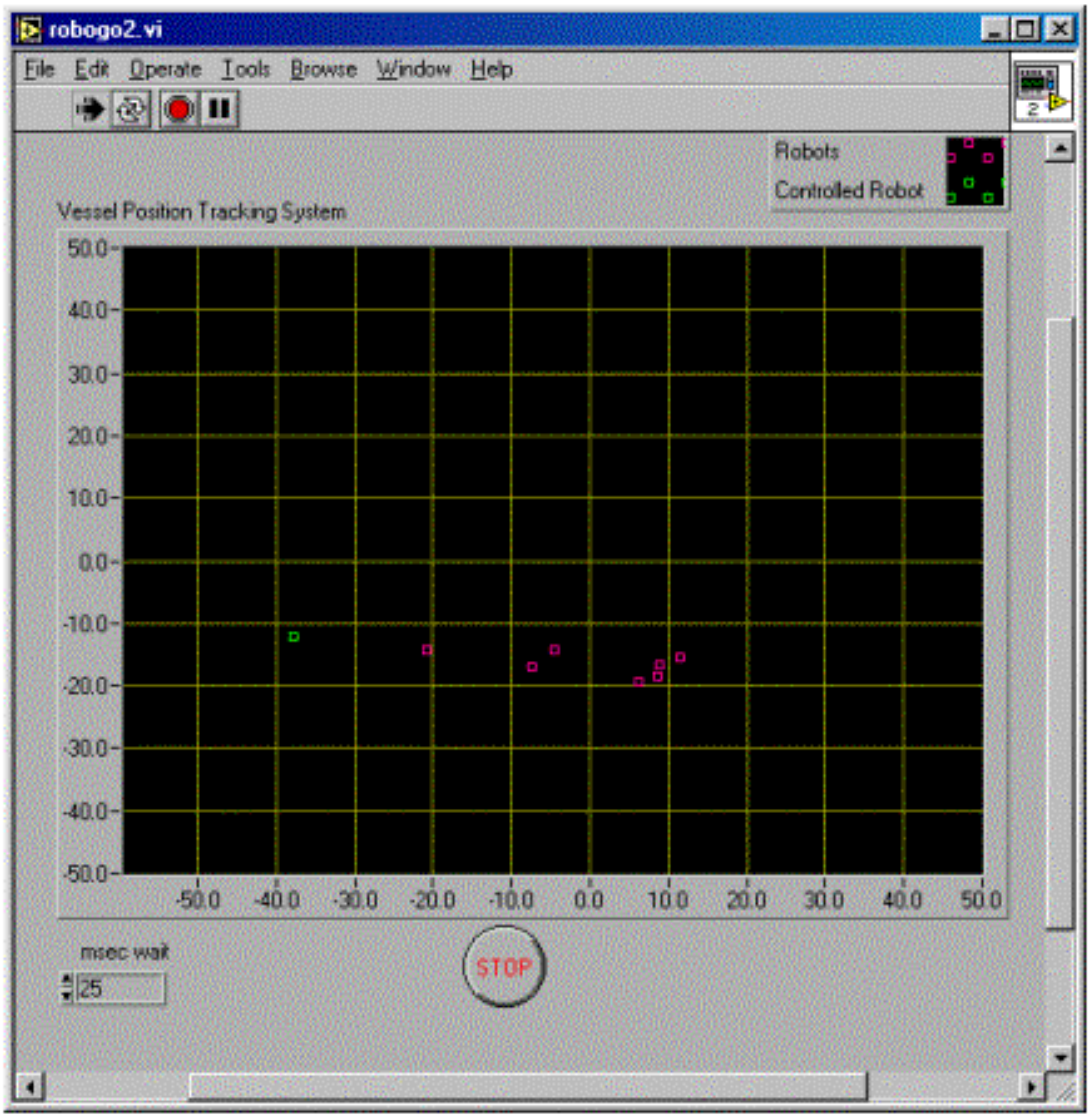

Figure 3.15. A tree formation heading across the screen from right to left. Notice that the proximity of the robots is closer at each successive level.

3.6 Summary

This chapter presented the preliminary design for a robot architecture whose main purpose is to give a robot the ability to learn about new situations that it encounters in its environment. The architecture described generates a neural network to create a behavior in order to cope with new situations encountered. This chapter detailed the work done to ensure that neural networks would be sufficient as controllers for the target test application of formation maneuvering. Once it was established that FFNNs were sufficient, the next step in the process was to test them in physical world. That is the subject of the next chapter. It describes the work done to transfer the ides and concepts described in this chapter to the robots in the lab.

Chapter 4. Mobile Robot Formation Maneuvering

Once it was shown that simulated robots were capable of formation maneuvering using feed forward neural networks, as detailed in chapter 3, the next logical step was to transition the work to the physical world. Although it may have been more exciting to jump immediately to testing with underwater vehicles, an incremental approach was taken; hence, mobile robots using a relative acoustic navigation system were used to simulate UUVs. This chapter opens by discussing the rationale for using mobile robots, and continues by describing the robots used, the sound system, and the many issues associated with it. It briefly describes the lab infrastructure that was created in order to do this research, and then continues with a discussion of the robot control algorithms. Finally, results and conclusions are presented.

The previous chapter went into some detail in describing the formation maneuvering simulation system. From that description it is apparent that while the simulation was successful in reaching its design goals, much more work would be required to make the modeling of sound and the dynamics of the vehicles, whether they are robots or UUVs approach reality. With that kind of computer simulation out of the scope of this research effort, and based on arguments by Brooks[6] and Webb[45] for the benefits of using physical systems instead of simulated worlds, it was decided that mobile robots would make an excellent, easy to use, easy to access, and economical to operate "dry simulation". Furthermore, since sound is a wave phenomenon, it has similar physical properties such as spreading and attenuation in air and water, which lends further merit to the decision.

4.1 A System Overview

This section provides a brief description of the robots used to study the formation maneuvering task as described conceptually in chapter 2, and specifically in chapter 3. The acoustic system used for relative navigation, and various issues, including the use of chirps, sound reflections in the lab environment, harmonics, etc. are discussed. For a more complete description of the robot hardware and its architecture see Roe[46].

Figure 4.1 below shows a picture of one of the three robots used in this work. The robot is an ActiveMedia Pioneer 2DX. The robots have a tricycle type wheel configuration, enabling them to turn in place. Their speed is variable from just over a crawl (about 10 cm/sec) to a top speed of about 1.8 meters per second. From the factory they come with bumpers for detecting collisions, a compass for determining orientation and ultrasonic sonar systems for detecting objects within a few meters. For these experiments, none of the factory navigation equipment was used. Instead, an acoustic system was developed with off the shelf parts. The next section describes this system and how it was used.

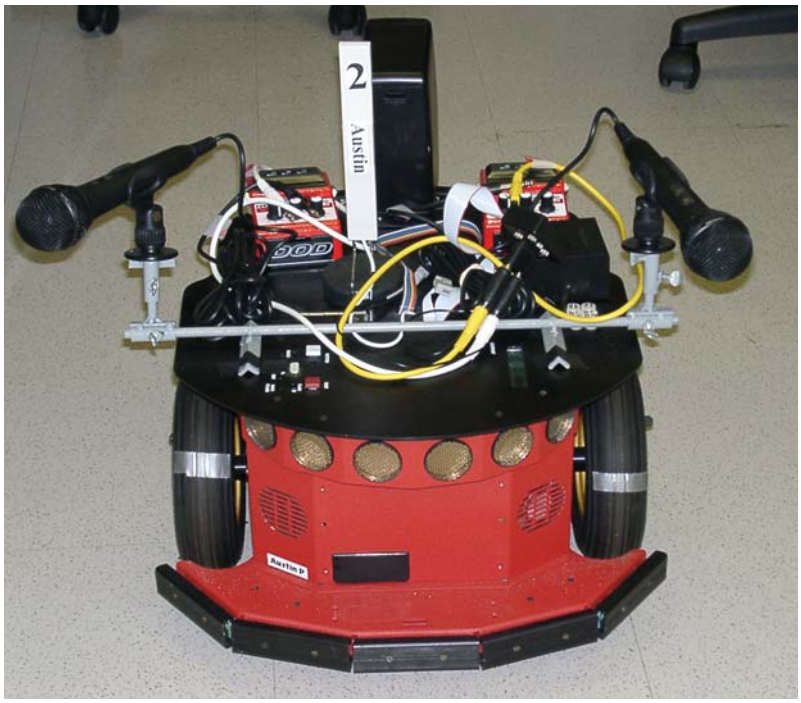

Figure 4.1. Pioneer 2DX robot with microphones and amplifiers mounted. Notice how the microphones are mounted on each side of robot, like ears. The orange boxes behind the microphones are battery powered amplifiers designed for electric guitars.

4.1.1 Sound System

The robot's sound system receives two channels of sound through a pair of voice type karaoke dynamic microphones whose signals are boosted by battery powered amplifiers designed for boosting signals from electric guitars. These inputs enter the robot's computer through its sound card's line in jack. The robot can transmit signals to other robots by using software to send signals to the sound cards speaker out jack. The speaker out jack is connected to a small battery powered speaker that has a built in amplifier and volume controls.

The software for the sound system has two main parts, the listening and chirping systems. They are standalone processes that run independently of each other. To get a feel for how the system works, consider a line of three robots. While robot number 1 is listneing for its leader at a typical sampling rate of 11025 Hz, robot number 0, it is also transmitting a 400 millisecond chirp once a second so that it can be tracked by robot 2. Robot 0, the formation leader is controlled by an operator or program using the labs desktop computers, so it does not need to listen for a leader. Robot 2 follows robot 1, so it listens for the chirps of robot 1, but since it is the last robot in the line it has no need to chirp. In the case when there are three robots in the line only robot 1 has to listen and chirp. If the line were longer all robots between the formation leader and the last robot would need to listen and chirp at the same time.

In the above example robots 0 and 1 are constantly chirping. In order for the robots who follow to discern who their leaders are, each leader robot chirps at a different frequency.

By chirping at different frequencies the follower robots can differentiate their assigned leaders from the others. Robot 1 only listens for the frequency in which robot 0 chirps, robot 2 only listens for the frequency in which robot 1 chirps and so on.

Direction of the leader is determined by assuming that the microphone with the loudest sound is closest to the source. If the signal level of the left microphone on a follower robot is twice that of the right microphone, it is assumed that the source is to the left, and vice versa. If the levels of the two microphones are about equal, then the source is assumed to be in front of the robot. In this system direction is based on the relative amplitudes of the received signals.

The next sections describe the operation of the sound system and how the robots use it for relative navigation. First the workings of the hardware and software that comprise the listening system is described. Next is a similar description for the chirping system. Finally, issues are discussed including problems associated with harmonics, artifacts in the chirps and similarities between acoustics in the air and water.

4.1.2 Robot Listening System

The listening system receives the acoustic data, consisting of chirps from the robots and ambient noise, from the microphones in the form of a sampled time series. The time series is then converted into signal strength in the frequency range of interest. This information is provided to the robots control program so it can determine which direction to steer.

Two methods that have been tested for determining signal strength at the microphones of the robots. They both calculate a running average of the signal strength in order to smooth out fluctuations caused by the chirp and environmental effects, but they differ in concept, implementation, and efficiency. The first method is an amplitude based system that considers only the signal strength in the frequency range of interest, while the second method is a matched filter based method that considers signal strength and wave form shape. Because the vast majority of this work was done using the first method, only it will be discussed.

The amplitude based method quantifies the amount of energy in a particular frequency range at each microphone. It works by continuously taking the power spectrum of the incoming time series data collected by the sound card from the robot's two microphones. Briefly, the power spectrum takes the FFT of the incoming signal, multiplies it by the complex conjugate of the FFT and divides the quantity by the number of points squared. This process converts the data from the time domain to the frequency domain. The LabView Auto Power Spectrum function is used to do this operation. Its output is a single sided power spectrum. Once in this form, the energy in the frequency range of the chirp is summed.

This system is similar to that used by Shah[31] and Kushleyev and Vohra[32] in that it is FFT based. Their system makes use of both amplitude and phase differences by comparing the processed sensor data to a data base of theoretical curves in order to determine the

angle to the sound source. Once the angle is found they use rule based logic to determine relevant parameters such as range and hemisphere. Both the of the FFT based systems are more computationally expensive the neural oscillator technique used in Reeve and Webb[33]. Reeve and Webb combine the detection and action networks and solve for the weights using a closed form solution. In practice they have to adjust their weights when they move from simulation to the lab. The work here keeps the two functions separate so that the learning algorithms can be used to train the neural networks instead of the researcher.

The listening process is repeated at a rate of four times a second. By doing this the values that the control system is working with remain current. The frequency resolution of the FFT is preserved by zero padding the input signal. This is useful for calibration, setup, and error checking of the system. The bins within the chirp range are summed and divided by the frequency width of the chirp. The resulting quantity is the average power of the chirp, in a range that is useful for calibration purposes. Values usually range between 10 and 150, making it easy to compare the microphone strengths in real time using screen outputs. A one second running average of this quantity is used as the processed sensor value. The running average helps smooth out the fluctuations that occur with acoustic data due tsoundo going from the source to the receiver in multiple-paths (multi_path). The process is shown in a block diagram in figure 4.2.

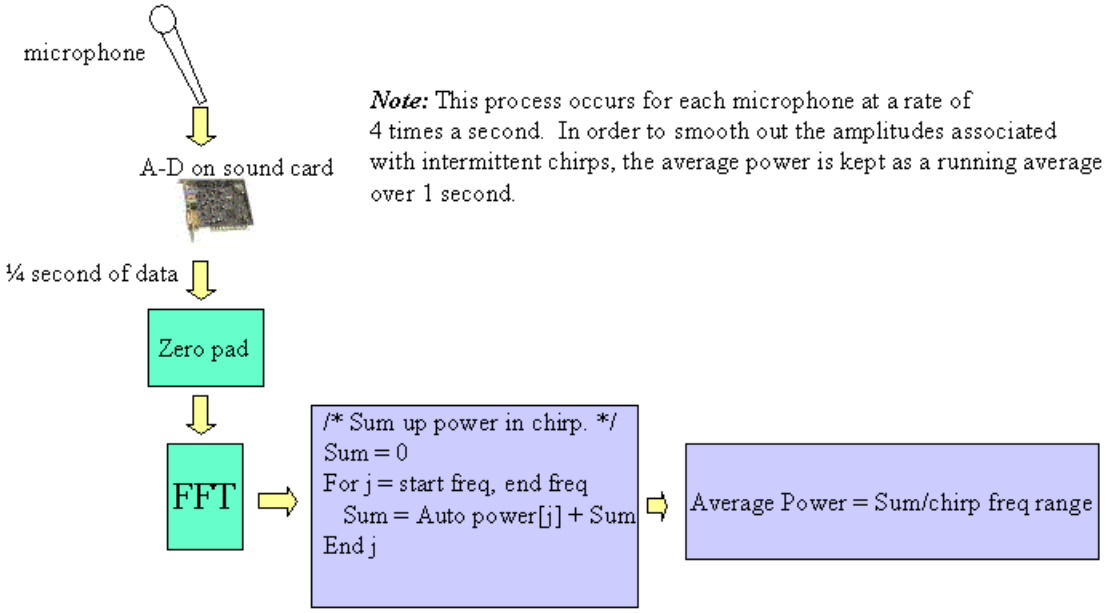

Figure 4.2. The raw microphone data is converted into an intensity value for a following robot's assigned frequency.

Testing has shown that this technique works well. It effectively measures the intensity of the chirp in the requested frequency range. In testing, it responds logically and predictably. That is, closer, louder chirps produce larger intensity numbers, and as they are toned down or moved away, the intensity numbers abate. It is robust to noise, including other chirps in other frequency ranges, 60hz noise from the labs lights, the robots fans motors, etc.

Since this method looks at only the intensity of the sound in the desired frequency range, sound that bounces off walls, ceilings, etc. is lumped in with sound that takes a direct path. This can cause problems, most notably when the lead robot is making a turn away from a wall, because the sound bounces off the wall and to the microphone on the follower that is closest to the wall. This reflection will fool the following robot into turning towards the wall instead of following the lead robot.

The next section describes robot's chirping system. It details why chirps were used instead of constant tones, and describes the waveform of a typical chirp.

4.1.3 Robot Chirping System

The communication system uses chirps in order to minimize problems associated with constant wave (CW) tones or pulses. Because the lab walls and floor are good sound reflectors, its acoustic characteristics are far from ideal. This could be similar to a highly reflective ocean environment such as shallow water or in a harbor with a rocky bottom. CW tones tend to saturate the air volume and create standing waves to the point that it is almost impossible for a person to audibly discern the location of the sound source. Constructive and destructive interference of the CW tone helps to create several regions of the room in which the sound is very intense, or barely audible. These local minima and maxima make tracking the sound an impossible proposition. Using a chirp helps to alleviate these problems in two ways. First, there is typically one chirp per second so there is time for the sound to attenuate. And second, the chirp sweeps from a starting frequency to an ending frequency so there is less time for the energy at any particular frequency to saturate the room and set up a standing wave, so the creation of regions of low and high intensity sound are reduced.

In practice the difference between using a chirp and a CW tone was found to be substantial. In tests conducted for this research, the level of difficulty associated with the CW tone was such that none of the algorithms would work using it. It may be possible to use a CW tone, but in light of the success of the chirp, further work with CW tones was not pursued. Shah[31] reports similar difficulties with pure tones and narrow band signals.

Initial testing of the hardware/software combination showed that the equipment had a usable frequency range from 300Hz to about 4000Hz. The bandwidth was limited by the response of the speakers, microphone, amplifier, and soundcard combination. The speakers cannot effectively and accurately radiate sound above 4000Hz and our microphone/amplifier/soundcard combination does not work below 300Hz. Later testing has shown that there is more usable bandwidth, but since that testing was done after the majority of this work was accomplished, the original range was used throughout this work.

Starting and ending frequency selection of the chirp is based on a balance of several issues, including the available bandwidth and the number of robots, the directionality of the speakers and the microphones and the existence of noise and harmonics. Early empirical experimentation showed that it was preferable to have the chirps as close in

frequency as possible, so that the attenuation and spreading properties of the chirps from robot to robot were nearly the same. That being said, because of the presence of a strong 2nd harmonic, the chirps had to be placed out of the range of other chirp's harmonics. For example, the 2nd harmonic generated from the 300 to 500 Hz chirp ranges from 600 to 1000Hz. In order to avoid this harmonic the 2nd robot's chirp was started at 1200 Hz giving it a 200 Hz buffer.

Typically the robots chirp for 400 milliseconds. The chirp is repeated once a second for the duration of the maneuvering session. Some experimentation has been done with shorter duration chirps repeated more often, but no conclusions have been made.

The next section discusses some of the issues associated with the acoustic system. It details some of the problems caused by the harmonics and addresses some problems associated with chirp waveform generation. Finally and most importantly it presents some sound measurement and model data that support the air/water analogy.

4.1.4 Sound System Issues

In transferring the simulation work detailed in chapter 3 to the lab several physical-world related issues needed to be addressed. One of the most relevant issues to indoor operations is the effect of harmonics. Another important subject is the structure of the chirp. But crucial to the whole concept is the validity of the air/water analogy. If the air water analogy is not valid then none of the work with the robots will be applicable to in water operations. The following paragraphs discuss these issues.

Experiments using the robot listening system detected a strong 2nd harmonic at all chirp frequencies. Because it was so strong, it could easily be lumped into any chirps that are at its frequency. The solution, as alluded to earlier, is to arrange the chirp frequencies of the robots so that the harmonics never coincide with legitimate chirps. With just three robots and a usable frequency range from 300 to 4000 Hz arranging the chirp frequencies is trivial. But each robot that gets added to the formation makes arranging the chirps harder. Table 4.3 below shows a chirp strategy for avoiding the harmonics.

Table 4.3. An arrangement of chirps in the usable frequency range of 300 to 4000 Hz. Each chirp is 200Hz wide and there is 200Hz separating the chirps from each other or harmonics.

Chirp Start (Hz)	Chirp End (Hz)	2nd Harmonic Start (Hz)	2nd Harmonic End (Hz)
300	500	600	1000
1200	1400	2400	2800
1600	1800	3200	3600
2000	2200	4000	4400

As can be seen from the table, the usable frequency space gets taken up quickly because of the harmonics. Table 4.4 below shows an ideal situation with no harmonics.

Table 4.4. An arrangement of chirps in the usable frequency range of 300 to 4000 Hz. Each chirp is 200Hz wide and there is 200Hz separating the chirps from each other. Unlike the previous table there are no harmonics so more chirps can be fit into the usable frequency range.

Chirp Start (Hz)	Chirp End (Hz)
300	500
700	900
1100	1300
1500	1700
1900	2100
2300	2500
2700	2900
3100	3300
3500	3700

As can be seen from Table 4.4 if the harmonics did not exist, more robots could use the frequency range. If the harmonics could not be avoided, expanding the frequency range helps, but the real solution is to remove the harmonics. While this is an important topic, and improving the chirp waveform eventually helped the problem, it is not the focus of this research.

Initial versions of the chirp program constructed a chirp that swept from a starting frequency to an ending frequency at a given an amplitude and duration. When the sound card transmitted the chirp's waveform, there would be audible "bumps" or artifacts at the beginning and end of the chirp, and as the frequency increased sweeping high frequency artifacts became more and more prevalent. These artifacts were more pronounced on some computers than others, but always present. Using the listening program it was clear that the chirp's energy was in the desired frequency range, but the artifacts were also present. A major contributor to the high frequency artifacts was an error in floating point to integer conversion in the waveform generation process. The original chirps were made with unsigned characters with no biasing, so the lower portion of their waveforms was being truncated, causing artifacts.

The audible "bump" problem was eventually cured by modifying the beginning and end of the chirps so that the amplitude ramped up to the desired value at the beginning and ramped down to zero at the end. The bumping problem was ultimately determined to be a consequence of speaker physics. The original chirp signal was constructed so that it started at zero, but this fails to consider Newton's second law. To simplify the explanation, consider the simpler case of a sinusoidal signal starting at zero. Although the initial value of the voltage being sent to the speaker is zero the derivative of this signal is not (recall that the derivative of a sine wave is a sine wave shifted by 90 degrees). The derivative of the signal corresponds to the velocity of the speaker cone mass, which cannot change instantaneously without an infinite force. The consequence of the original signal design is that we were expecting the speaker cone mass to

instantaneously go from zero velocity (stationary) to maximum velocity, and this resulted in a very audible bump.

Aside from the severe range limitation in the lab two other interesting phenomena were observed during testing with the robots. Reflections from walls were seen to be especially troublesome because they increase the acoustic energy that arrives at the microphone nearest to the wall. If a leader/follower pair are traveling parallel to and too closely to the wall the follower will tend to head to the wall because the wall reflects the sound energy that would normally radiate into the air away from the follower. Figure 4.5 illustrates this phenomenon. Potential solutions to this issue include communication of the leaders intentions (so the follower robots know that the leader has not turned) and the use of time difference of arrival at the two microphones. This effect also has an adverse effect on the training and calibration of the controllers if this is done with the robot too close to a wall.

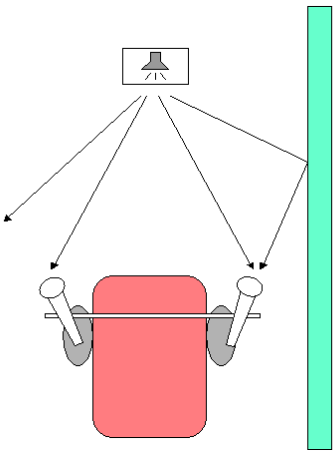

Figure 4.5. In this figure the robot (the red rectangular figure) is heading parallel to the lab wall (the thin green rectangle to the right of the robot). The speaker is in front of the robot radiating acoustic energy. Because the wall is so much more dense than the air, it reflects a large portion of the sound energy back into the room. In this case, the wall is reflecting sound back to the robot that would normally be radiated elsewhere. This causes the robot to read a higher intensity value on its right side, causing it to errantly turn towards the wall.

Another interesting situation occurs when the robots are near the wall and the lead robot turns abruptly away from the wall, causing the following robot to head to the wall. This happens because there is no longer a direct acoustic path between the leader and the follower and the strongest signal is that reflected off of the wall. This situation happens only occasionally and is dependent upon the distance of the robots from the wall, between each other and the rate of turn of the lead robot.

Because of the similarity with which sound propagates in air and water, we expect similar problems to arise in ocean environments, especially near ships, or in harbors with rocky bottoms.

4.1.5 Validity of the Air/Water Analogy

In order to better understand and quantify the performance of the amplitude differencing approach and to find out if using an acoustic communication system in the air was analogous to using a similar system underwater, an analysis and computer simulations were undertaken. Gausland[47] indicates that while sound has similar characteristics in air and water, the difference in density of the two media results in some differences in the amount of energy required to make sound waves and in the different types of waves that the mediums can support. If the air/water analogy is not valid, then none of the work with the robots will be applicable to in water operations. The results of this analysis were compared to a series of sound measurements done in the lab and in a quiet non-reflective outdoor environment. The following paragraphs detail this work. First a simple medium independent spreading model is presented. Next, a more complex model is compared with measurements for both air and water. Following that range vs. intensity at different frequencies is discussed for both indoor and outdoor environments, and finally intensity vs. receiver angle in an indoor setting is analyzed.

Figure 4.6 shows the theoretical decrease in power with range of an acoustic signal due to spreading. The blue curve is for cylindrical spreading, such as you would expect in a shallow water environment where the bottom and the surface are reflective boundaries, and the red curve is for spherical spreading that you would see in deep water. For both environments, the signal strength falls off rapidly at close ranges in accordance with the square law. The equations[48] are shown below.

TLS[R]	Transmission loss at range R due to spherical spreading
TLC[R]	Transmission loss at range R due to cylindrical spreading

R	Range of receiver from sound source
R0	Reference range (usually 1 meter)
S	Intensity of sound at reference range
I[R]	Sound intensity at range R

For spherical spreading:
$TLS[R] = 20*\log(R/R0)$
$I[R] = S - TLS[R]$

For cylindrical spreading:
$TLC[R] = 10*\log(R/R0)$
$I[R] = S - TLC[R]$

For the current work we are sensing direction by measuring the power difference between the microphones that occurs due to the power drop across the distance between them, similar to the inter level difference algorithm described in Shah[31]. For greater ranges, where there would be little power difference between the microphones, other techniques will have to be used that take advantage of the difference in time of arrival or microphone directivity.

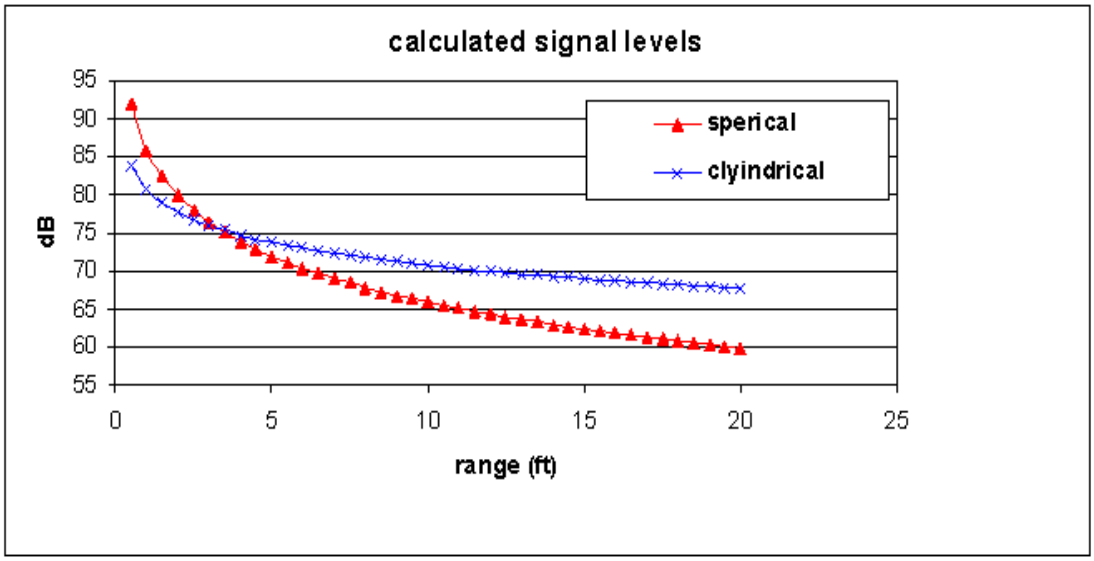

Figure 4.6. This figure shows the calculated signal loss due to the spreading of acoustic energy. Note that these calculations are medium independent, that is they are applicable to both water and air.

A detailed acoustic model, a version of the U.S. Navy Standard PE (Parabolic Equation), was used to model both outdoor air and shallow water environments. This model includes the effects of both spreading and attenuation. The results for the in air model are compared with data of the same frequency, collected using a sound pressure meter at different ranges, shown in figure 4.7 below. This figure shows close agreement between what the model predicted and the measured data. The short range dip in the model data is an artifact due to model initialization parameters.

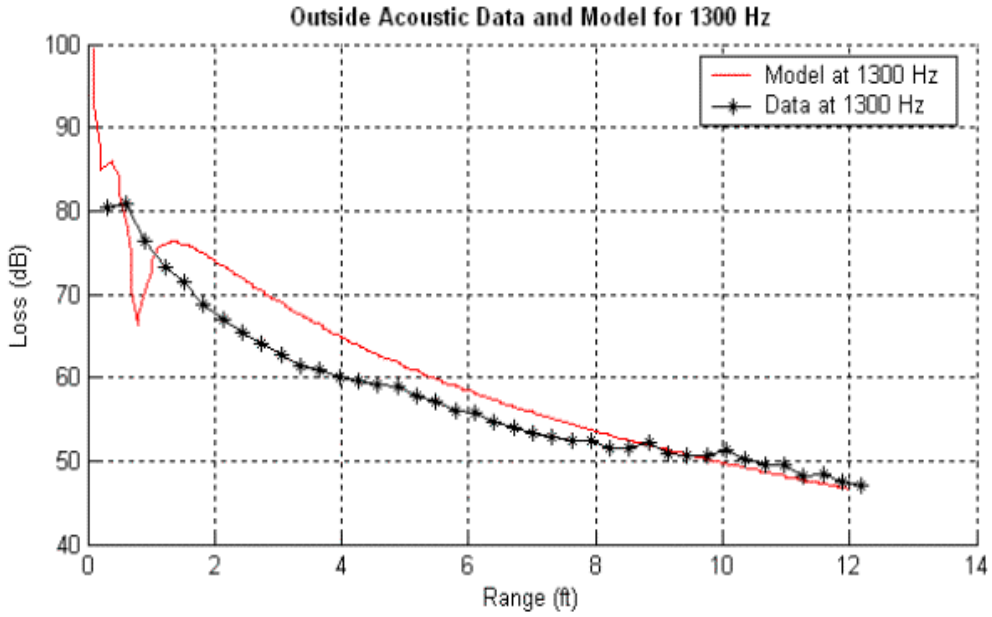

Figure 4.7. Comparison of a typical acoustic data set collected in a quiet outdoor environment and the acoustic model results for the same frequency. Background noise level for the collected data is 51 dB, the chirp frequency range is 1200 to 1400 Hz.

The Pekeris waveguide[49] is considered a reasonable representation of a simple underwater environment and is often the first step in modeling the complexity of the ocean acoustic environment. The waveguide is represented by range independent bathymetry, isovelocity water column sound speed, and a bottom as an infinite fluid halfspace. The plots in figure 4.8 use the standard inputs for the Pekeris problem with modifications to the input frequencies. This should give an indication of how the transmitted signal will be affected by an underwater environment. Comparing this result with the earlier plots suggests that the affect of water will be similar to that of air, indicating that the decrease in signal power with range will act similarly. The oscillations seen clearly at longer ranges in the model curve are due to interference patterns. These oscillations are less visible in the measured data due to the sampling interval.

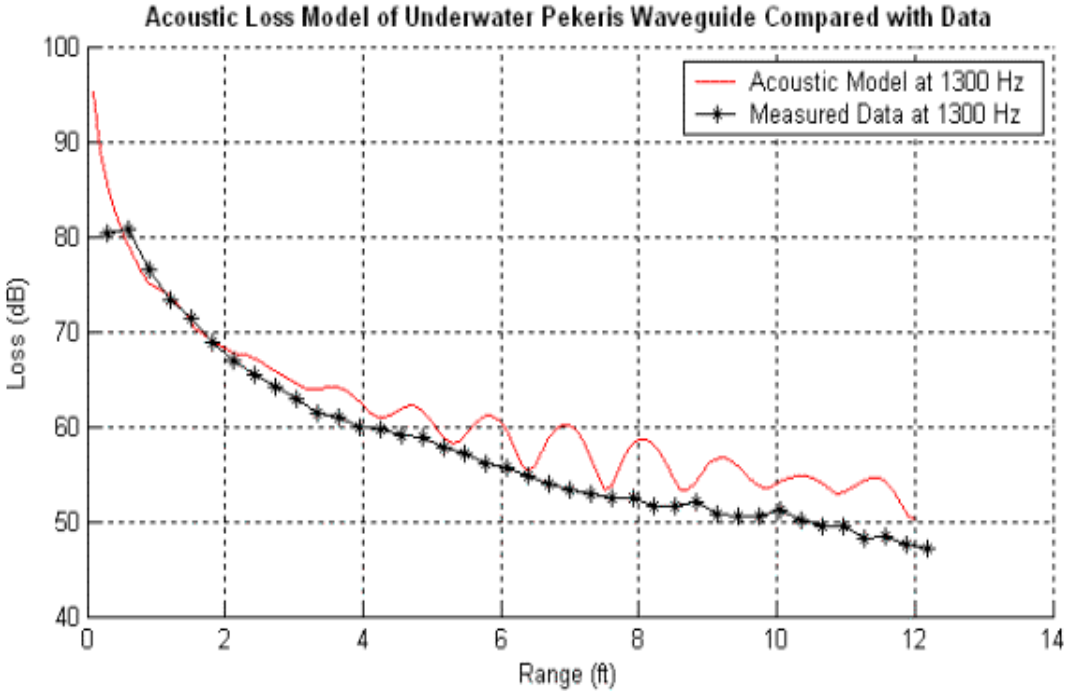

Figure 4.8. Comparison of the collected data and the acoustic model results using the Perkeris parameters for an underwater environment using a source frequency of 1300Hz.

Having established that the air/water analogy has some validity, in order to compare the differences between the indoor lab environment and a less reflective outdoor environment a series of pressure versus range tests were run. The results of the outdoor tests, where the dominant reflective surface is the ground, is shown in Figure 4.9. The shape of the curves match that predicted by theory and simulation, and this figure also shows the marked affect that frequency has on signal attenuation. For this environment the effective range for the higher frequency is expected to be about 10 feet.

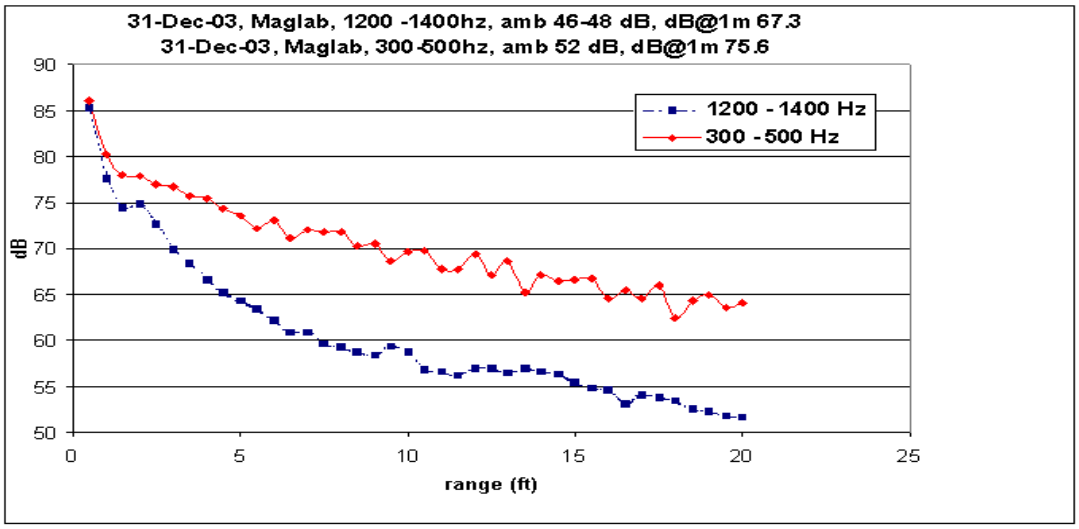

Figure 4.9. This figure shows an intensity/distance plot of measurements taken in a quiet non-reflective outdoor environment. Notice how the intensity of the sound in the 1200 to 1400 Hz range drops predictably in the 0 – 10 feet range.

Results from the indoor range test using a 300 – 500 Hz chirp are shown in figure 4.10. As seen in plot, once the sound meter is more than 4 feet from the source, the sound pressure levels off to the point where there is no measurable power difference between the two microphones. In this figure we also see that the sound intensity change with range deviates from the theoretical predictions for simple spherical and cylindrical spreading. The sound intensity is seen to increase or decrease unpredictably with an increase in range. This is due to the acoustic reflectivity of the walls and the formation of standing waves.

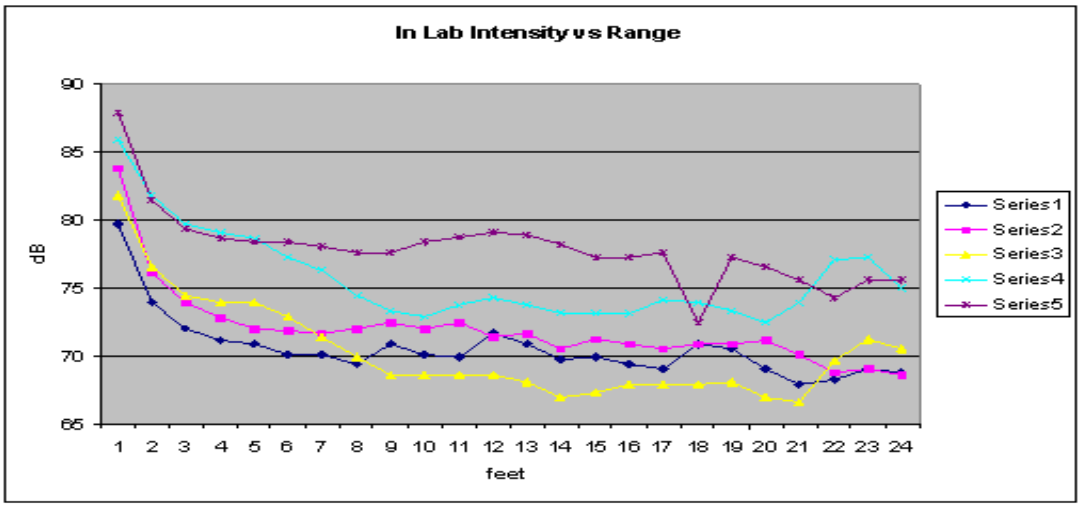

Figure 4.10. In this figure the data from several in lab intensity versus range experiments is plotted. The chirp is in the 300 to 500 Hz. range. The different curves represent different orientations of the speaker and height of the speaker receiver pair. None of the curves shows a consistent amplitude drop after 4 feet from the receiver. Background noise level is 46 dB.

The data in figures 4.9 and 4.10 suggest that the amplitude differencing method will work in both environments, but the reflectivity of the walls in the lab greatly limits the effective range. The results of figure 4.10 suggest that once the receiver is much beyond 4 feet from the source, multi-path sound, i.e. sound that gets to the meter from some other path than a direct path, is causing the readings to become unpredictable. In order to test this hypothesis, a sound intensity versus angle test was run in the lab at a distance of 12 feet from the sound source. The results, shown in figure 4.11, indicate that at this range, there is little discernable difference in the sound intensity no matter which way the receiver is oriented. This effect, caused by the multi-path as mentioned above, makes it nearly impossible to discern the direction of the sound source.

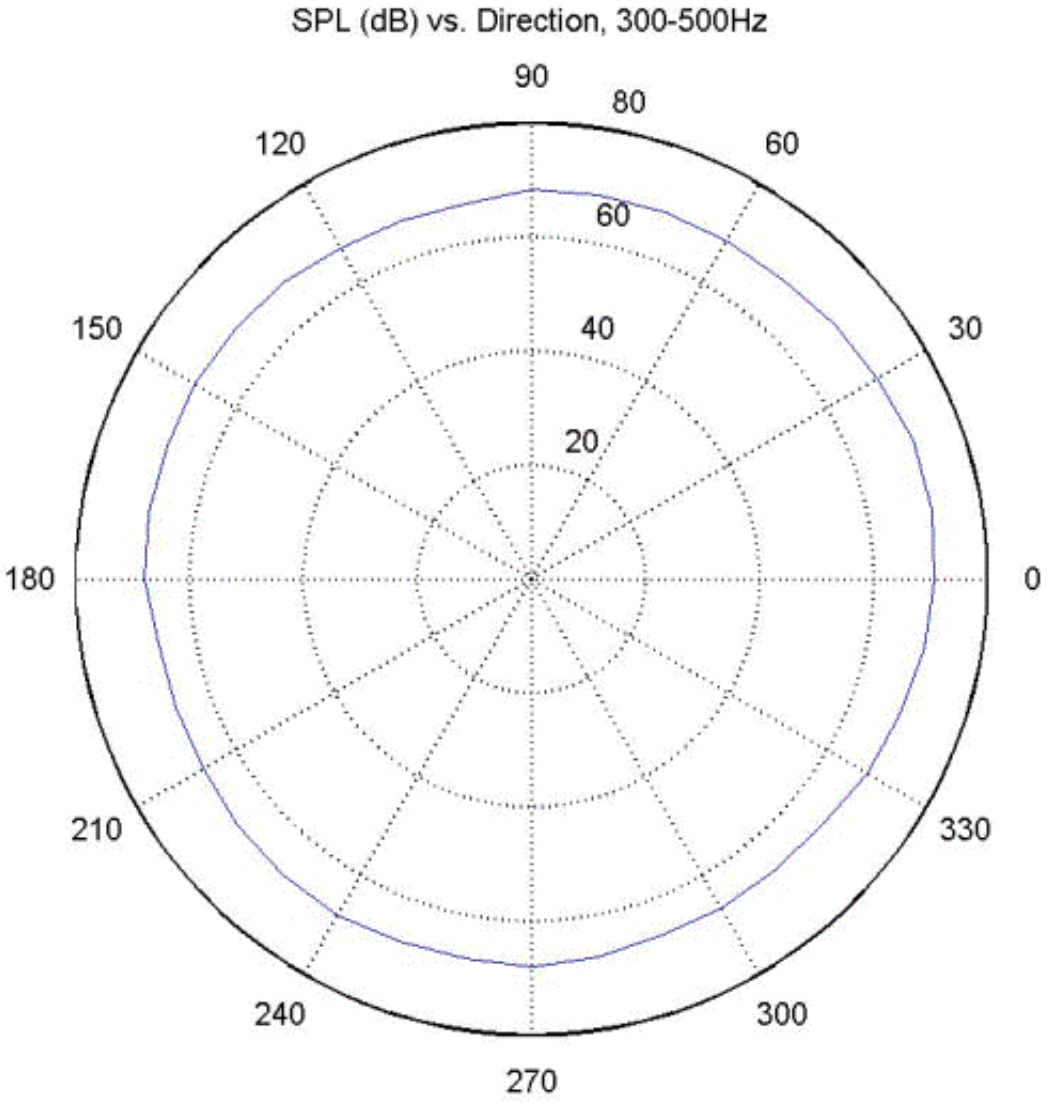

Figure 4.11. This figure shows the sound intensity versus angle data. Note that it is almost the same no matter what direction the meter is facing.

To summarize, the model data and collected data show that the amplitude differencing technique clearly works, but has limitations in highly reflective environments. Testing by Researchers[31] [32] indicated similar results and difficulties, particularly with noise and reflections of sound, in their testing of similar amplitude difference techniques. It also shows that working with acoustic systems in the air is a valid step to take towards the goal of using similar systems underwater. The next section briefly describes the control methods tested. It describes the controllers but detailed descriptions of the learning methods that produced them are left to chapters 5 and 6.

4.2 Control Methods

Three methods of control have been tested. They are a classic logic approach, a behavioral approach, and a neural network approach. Typically the tests consist of an operator guided formation leader robot and one or two following robots using one of the control methods discussed in this section. The formation leader is guided by the lab's in house control system. For more detail on this system see Roe[46]. The classic logic approach is used in this work as a baseline method to provide a performance comparison with more advanced control strategies. The behavioral approach was implemented to contend with situational ambiguities that the reactive nature of the classic logic approach could not handle. The neural network controller was pursued since this type of controller is better able to contend with nonlinear sensor data and has the potential to serve as the core for a learning architecture. The following sections provide more detail.

4.2.1 Classic Logic Approach

As a baseline method to be used for testing, the robots were first programmed using simple logic that closely modeled that of Braitenberg vehicles[10]. Braitenberg vehicles typically use light sensors to directly control motors on wheeled vehicles. Various wire arrangements allow the vehicles to seek light or avoid light. For this work, sound sources and microphones were used instead of light sources and light sensors. The basics of the classic logic method are as follows:

```
While (running) {
        If (either of microphones reads a very loud intensity) then
                Velocity = Stop;
        Else
                Velocity = Go;
        If  (microphones read intensities that are close to equal) then
                Direction = straight;
        Else
                If (right microphone value is greater than left) then
                        Direction = right;
                Else
                        Direction = left.
} /* End while. */
```

The first if statement keeps a robot from colliding with its leader. The second statement creates an "on center zone" so that the robot will go straight. This statement is not as

important in the simulator because reflections, bounces and noise are not modeled, but without it in the lab the robots will always turn left or right, creating a serpentine path. The final if statement selects the direction to turn, given that the microphone intensity was sufficiently different enough to not fall in the on center zone. Note that this is a strictly reactive control scheme in that it only reacts to the current sensor values. One implication of this control method is that the robot will not be able to discern if it is heading directly towards or away from the source. The next section describes a method used to solve this problem.

4.2.2 Behavior Approach

The Behavior approach was intended to solve some of the problems seen with the strictly reactive approach by considering the current and past sensor values. Examples include allowing the robot to determine if it is getting closer to the source over time, or scanning to determine the most likely direction of the leader when it is at farther ranges.

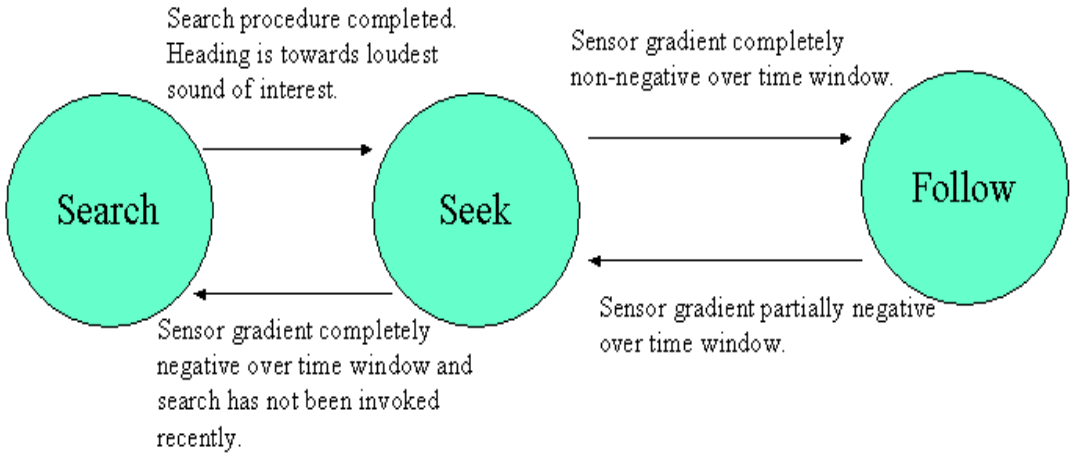

Figure 4.12. The 3 behaviors and how control is transferred between them. Note that the sensor gradient over time regulates all the transitions except that of Search to Seek. After the Search function is complete, control automatically transfers to Seek. A timer keeps the control from returning to Search for a tunable time period.

As shown in figure 4.12, three behaviors are used to find and follow the sound source. They are follow, seek, and search. The follow mode is for maintaining a desired distance behind a lead robot, the seek mode is used when the robot knows the direction to the lead robot but is too far away for the follow mode, and the search mode is used to determine the direction of the lead robot. The selection of the behavior is based on the gradient history of microphone intensities. Much like subsumption only one behavior is active at a time, with the added enhancement that some behaviors require a waiting period before they become active again.

When directly behind its leader, a robot is normally in follow mode, which uses the same basic logic as the classic logic module. As the distance from the leader increases it is difficult for the robot to discern the direction to the leader because the difference in power between the two microphones lessens as seen in the previous discussion on

acoustics. In this case we want the robot to have more freedom to maneuver left and right to help find the direction to the leader. If the distance between the two robots is too large (i.e. acoustic signal is too weak) or grows over time the seek behavior then becomes active. Seek allows the robot to respond more quickly to directional changes by varying the "*close*" parameter as a function of sound intensity, while follow tends to keep the robot moving straighter.

If the robot realizes that it has lost track of the source by detecting a negative sensor gradient, search acts to reorient it towards the peak level of the sound source. This is accomplished by rotating the robot and taking advantage of the directivity of the microphones to localize the source. This procedure is similar to that detailed by other researchers[32] when determining if the sound source is in front of the robot or behind, except that the angle to the source is never directly calculated. Here the robot rotates until the source is in front of it, as opposed to making a rotation, solving for the angle to the source, and then continuing in that direction. The behavior algorithm will work with a moving source, while the previously mentioned algorithm[32] will not. Once oriented in the general direction of the source, the robot returns to seek mode, and is prevented from entering search for a specified period of time.

4.2.3 Neural Network

The neural network controller, shown in figure 4.13, is based on a feed forward neural network with one hidden layer trained by a genetic algorithm (GA). As described in chapter 3, it was shown[50] that this concept has merit when applied to formation maneuvering. Recall that the earlier network's output was a speed and heading adjustment. Since the robot is a physical system, it was decided that for the initial tests it would be best to let the operator retain control of the speed adjustments. Along the same lines, the networks used on the robot do not control the amount that it turns. They only indicate the direction of the turn, i.e. left, right, or no turn (straight). The degrees per turn is an operator adjustable parameter. For larger turns the robot repeats the turn command until it is oriented in the manner it desires.

Although similar to the work in chapter 3 this work has some distinctive differences. In the previously mentioned work, the neural network was trained by repeatedly letting a simulated robot controlled by the network follow a computer controlled simulated robot that made random course changes. The fitness function in the GA optimized the distance between the leader and follower. After several generations, a controller was "grown" that could keep a follower robot consistently close to a leader robot. While this approach is ideal for a simulator, physically executing the generations of runs to develop a good controller on the lab robots was not an option due to time and wear and tear on the systems.

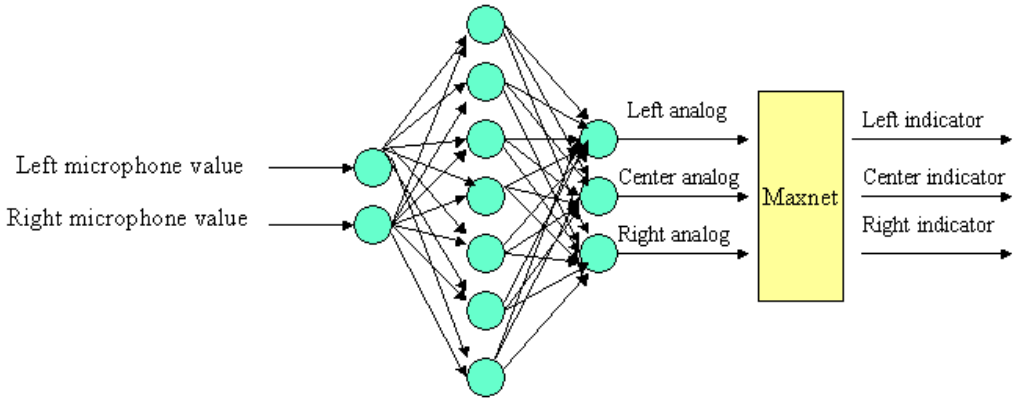

Figure 4.13. This figure shows a typical neural net used to control the robots. The inputs consist of the conditioned microphone values coming from the listening program. There is one hidden layer and the output layer has 3 nodes, one for each decision that can be made. Since the robot commands, left, right and forward (center) can only be done one at a time, a winner take all scheme (maxnet) is used to condition the outputs.

To avoid running physical tests, the robot was trained to estimate where a sound source was in relation to the direction the robot was currently facing. The robot was allowed only three possible choices: the sound is either to the left of the robot, in front of it, or to the right. The training set for the neural network was developed by collecting microphone data for each of these three sectors. This was accomplished by telling the training program what sector the source is in and then by moving the source to various locations in that sector during the data collection process; this process was repeated for each sector. The GA fitness function was designed to maximize the correct matches between the output of the network and the training data. Once the robot can guess the direction of the sound source, that information is then used to guide the robot towards the source.

The FFNN used to control the robots is essentially a neural network version of the classic logic system, with the exception that the training process determines the size of the on-center zone instead of the operator. This approach allows straightforward compensation for variations in robot characteristics such as voltage levels, leader frequency, microphone response, and amplifier characteristics, etc. This approach of growing a custom network for each robot helps alleviate the need to calibrate each system. However, if the conditions for which the training occurred change markedly, the resulting network is not suitable. In this case a new network would need to be grown.

In summary, the robots were tested with three very different methods of control. The classic logic approach and neural network methods worked well (results of the simulation are presented in the next chapter) in both simulation and in the lab environment, allowing line formations of the robots to run laps around the lab, while the behavior algorithm did not fair as well in the lab. The next section provides some detail on testing issues and some snapshots of tests run in the lab using the robots.

4.3 Experimental Results

This section begins by discussing issues addressed during initial testing and presents snapshots taken from video of the robots executing a line formation in the lab. Although the robots have wheel revolution counters that can be used for positioning estimates, their error rate[46] is too high for tracking the robots. A better positioning system using a camera based technique[46] is under development but was not ready for use for these tests.

Initial testing was done using input data from the listening system that was scaled so that its range was between –1 and 1. Using this method line formations that were able to run laps around the lab were successfully created. The FFNN controllers worked well as long they were within close range (about 1 to 3 feet) of their lead robots, but once out of that range they failed. Changing the inputs from scaled microphone values to relative microphone values largely alleviated this problem. The logic below outlines the process.

The variables are defined as follows:

left : the value the sound system returns for the left microphone.
right : the value the sound system returns for the right microphone.
Big : the greater of the left and right microphone values.

The algorithm is given below:

```
/* Get the larger of the two microphone values. */
If (left > right)
        Big = left
Else
        Big = right

/* Normalize the left and right values. */
left = left/Big
right = right/Big
```

This procedure removed the intensity information from the inputs and left only directionality information. When using the non-relative values, training examples from the same orientation, but a different range are very different from each other numerically. Using the relative values, they are closer to the same at the ranges used in these tests. The result was that the limited number of training points provided a better description of what the robot would experience during following maneuvers. This in turn helped to lower the unpredictability of the networks responses. With this change it is possible to create networks that operate smoother than either the classic logic or behavior methods.

Figure 4.14 shows a sequence of frames from the start of the formation to when the formation rounds the first turn of the rectangular grid laid out on the lab floor.

Figure 4.14. The lab robots in a line formation maneuvering in a square pattern on the lab floor. The lead robot does not have microphones, it is being controlled by one of the lab's computers. The two following robots are equipped with microphones.

Figure 4.15 below shows the robots rounding the curve and heading back towards the operators consoles.

Reeve and Webb[33] reported good results in seeking to a stationary sound source in their work investigating cricket phonotaxis. Instead of using FFTs and a neural network to isolate the chirp and control the robot, they used a neural oscillator that acted both as a band pass filter and a directional controller. Using this approach their robots seeked to a sound source emitting 20ms chirps in the 4.7kHz sound range. The weights were found by using a closed form solution for their neural model. Since their goal was to emulate the physiological structures and functioning of the cricket, their system is not designed with learning and changing environment in mind. That being said, it does show that neural networks are effective controllers, and does suggest that with a system of tuning the neural oscillators, it may possible to replace the FFT process, resulting in a savings of computational resources.

Up to eight laps were run in formation using both the classic logic approach and neural network controllers. Testing of the line formations in an outdoor environment has not yet been done, but from the results of the outdoor sound tests, it is anticipated that the performance will be better than that in the lab.

Kushleyev and Vohra[32] developed a system to localize sound based on a two sensor system modeled after a human head. They show good results for a simulated robot

tracking a moving sound source. As expected, results for experiments without noise added are better than those with noise. Their algorithm is rule based and uses an FFT based process to determine range and distance to the target. Their testing indicates that they too experienced many problems with noise and reflections in their lab. Since their system is rule based it does not lend itself to changing environments. They do not show results with multiple robots in simulation or in the lab.

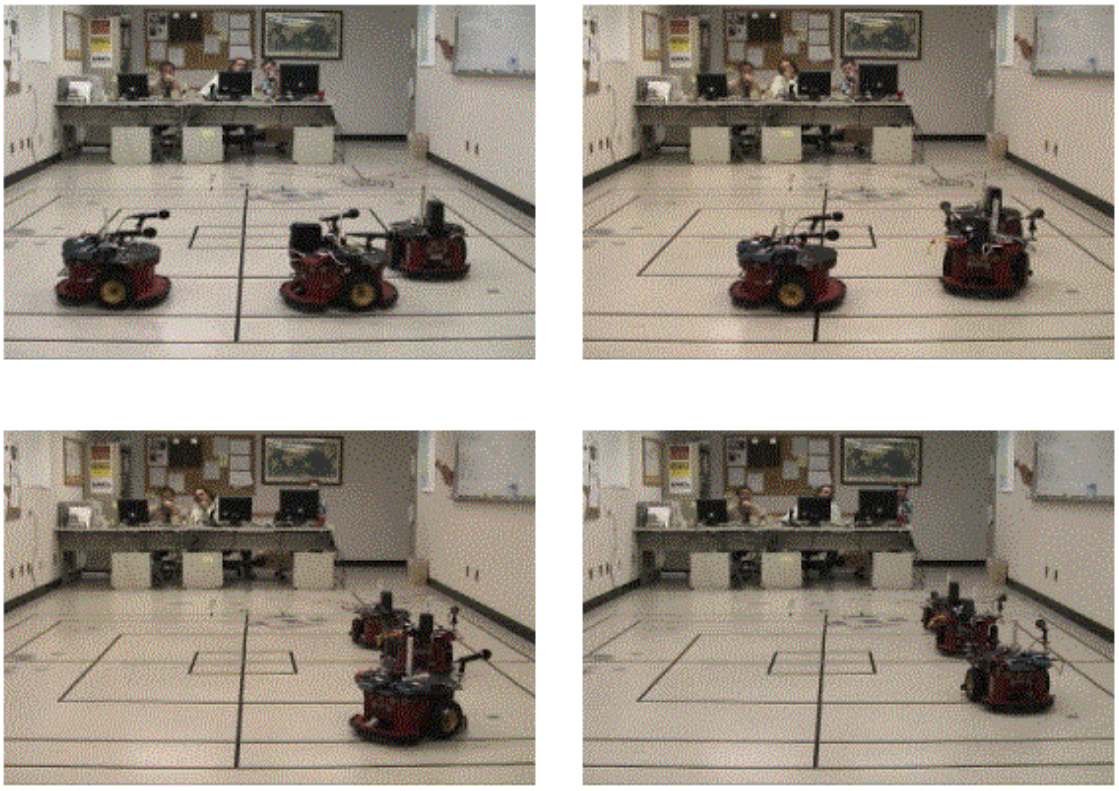

Figure 4.15. The lab robots continuing their line formation from figure 4.14.

4.4 Summary

This chapter opened by describing the mobile robots and discussed why using them to study UUVs had merit. The hardware and software that comprises the robot's acoustic communication system was described, along with various issues. The chapter closed by presenting the three robot control methods that were tested on the physical systems in the lab; classic logic, behavior, and neural networks. The human-in-the-loop supervised method of training the neural network was described. The next chapter describes the initial work done to automate the training process of the neural networks.

Chapter 5. Learning Using Sensor/Action Pairs

In chapter four, three control methods were discussed. Each of them had their strengths and weaknesses, but because it is a more straightforward process to develop a program that produces usable neural networks than to create programs that write programs using symbolic methods (coding of algorithms in a language such as C or LISP) efforts have been focused primarily on neural networks. As stated earlier, writing programs that produce other running programs at the very least involves ensuring that the machine produced programs that are syntactically correct and cannot do harm to the computer system. Using a neural network avoids these issues. In the previous chapter the neural networks were trained using a human-in-the-loop supervisory system. This chapter details efforts to automate this process using a supervised learning strategy. It starts with a short introduction and continues with a discussion of recent memories, how they are used, and their content. Next, reactive learning is detailed, followed by non-reactive learning, and finally a presentation of test results.

5.1 Introduction

Automation of the training process is important because for a robot to be effective in remote environments, or work in conditions that change often, it needs to be able to adapt without the help of humans. Said another way, a neural network that was created using the original training data could become ineffective over time, i.e. in order to keep the neural network up to date, it may become necessary to compensate for changing environments and sensor characteristics. In a similar work, Rucci et al used a type of reinforcement learning, which they called value-dependent learning, in their development of a robotic barn owl[34]. Their system could adjust itself when environmental changes occurred. One of the main differences between their work and the work presented here is that one of the primary goals in this work is to reduce the physical repetition required by the robot during the learning process.

Consider an example in which a robot using a relative acoustic navigation system following another robot moves into an area in which the characteristics of sound change because of temperature or humidity. Or perhaps the environment remains constant but over time the sensors get partially obstructed causing them to report values differently from before. In order to address these problems, a method of training from recent memories was developed. The use of recent memories helps the robot minimize physically testing alternate solutions, thus saving time and wear and tear on the robot. It also makes a-priori knowledge less important because much of that information is contained in the recent memory.

For this method to be successful, some assumptions are required:

1. A goal must be specifiable in terms of sensor values.
2. The system must be able to observe its progress in order to detect when goals are not being met.

3. The data in the recent memory that the learning system is using must contain sufficient information to form a complete strategy for the situation at hand.
4. The learning system must be able to generate a new neural network before the environment changes enough to make it irrelevant.

The first assumption calls for goals specified in terms of sensor values. This helps to reduce the amount of a-priori knowledge required. The next assumption implies that the system has or will have an observer process that can compare measured progress against pre-defined goals. It is recognized that this is important, but for this research the focus has been on the learning process, not the observer, or the architecture that integrates these processes to provide a system that iteratively improves and adapts its controller to the environment. The learning process can only learn from the data that it is presented with. As the third assumption states, without enough information to form a complete strategy, or without a method to recognize this and a scheme to complete the information at hand, any strategy learned will be incomplete. The forth assumption requires that the new neural network be generated rapidly enough to be brought on line before the situation changes. It would seem that the sooner the network is generated the better, but there will always exist environments that change too quickly for this type of adaptation process. These environments may require a predictive process, which again is not the focus of this work.

The next section discusses the recent memories. The section starts by explaining why they are used, follows with their content, and then closes with a discussion of the different methods that have been used to collect them.

5.2 The Use of Recent Memories in Learning

Learning from recent memories is used so that the robot or agent can avoid regimes of directed testing of trial solutions. The idea is to record a memory consisting of sensor data/action pairs and learn from them what actions helped to optimize goals. By learning from what has already been experienced the robot can create a "draft" controller that can be iteratively improved as the robot operates in its environment in fulfillment of its goals. Of primary importance for this strategy to work is for the agent to be able to identify where in the recent memory the robot's goals were being realized. The next chapter details two algorithms that relax this constraint. In conjunction with being able to identify these parts of the memory, the memory must contain enough of usable examples for the robot to learn all the functions needed in order for it to realize its goals. If this information is not available, or cannot be generated from what is available, the robot will learn an incomplete strategy.

As stated earlier, the process is seen as iterative, so as time goes on the strategy will become more complete. In this research the robots architecture was incomplete, so the process could not be iterative. This required taking measures to ensure that the recent memory contains the necessary information. This section focuses on the acquisition of the recent memories. Later sections discuss the different processes used for identifying the parts of the recent memory that can be used for training.

When an agent or robot is starting the learning process from scratch, i.e. no a-priori knowledge, it needs some way to explore the environment so that it can learn. At this point in its development, it only knows it goals, but has not learned a sensor/action mapping that can direct it towards its goals. In order to collect information for the learning process, an environmental exploring method was developed from which a recent memory was made.

Using the formation maneuvering/following task an example, the robots, whether simulated or real, were driven by an operator in order to collect a recent memory. The operator would navigate the robot around the sound source making random course changes towards and away from it. In this way many examples that could be learned from would be recorded for the learning process. If more of one type of example got recorded than another a mirroring process, discussed in a later section, was used to generate a more complete data set.

In the interest of keeping with the original idea of creating a system that did not need to rely on humans, it was decided to automate the exploration of the environment/recent memory creation process. The goal was to have a system whose initial knowledge consisted of its goal, defined in terms of sensor values, and a baseline environmental exploration routine. Using these tools the system would begin by exploring its environment using the baseline exploration routine. A memory of the sensor/action pairs created from the environmental exploration would be created. This recent memory would in turn be passed to the learning system which in turn would be used to create a controller.

Three types of environment exploration controllers were tested. The first was a controller that mimicked the human operator by guiding the robot along a path with occasional random course changes. Testing with this controller made it apparent that the operator was not in fact driving randomly. Further analysis showed that operator directed "random" drives were actually drives in which the operator kept the robot near the source by guiding it on a path that took it towards the source from a variety of directions in a haphazard manner. The computer controlled random driving was truly random, often causing the robot to wander to far away from the source quickly. Since the random course changes usually summed to a near zero total change in course, once the robot passed the source it never returned. Because sound attenuates quickly with distance (see figure 5.1) once the robot got any appreciable distance away from the source there was not enough amplitude difference at the robot's microphones to provide useful information from which to learn.

The 2nd controller did not fair much better. It was based on a preprogrammed pattern around the source. It suffered from the same attenuation problems as the random controller. While it did have the advantage that it was easily repeatable, most of the tested pre-programmed patterns did not provide enough examples in ranges where amplitude differences at the microphones were great enough to be of use. It also suffered from the disadvantage of being tied closely to the target application; even if a good pattern had been found, the aim was to find a more general method of creating a good

initial recent memory that did not require a human operator to construct a specific learning scenario. Figure 5.2 below illustrates a pre-programmed pattern that was tested, but did not provide adequate information. The pattern is overlaid onto a range/intensity plot.

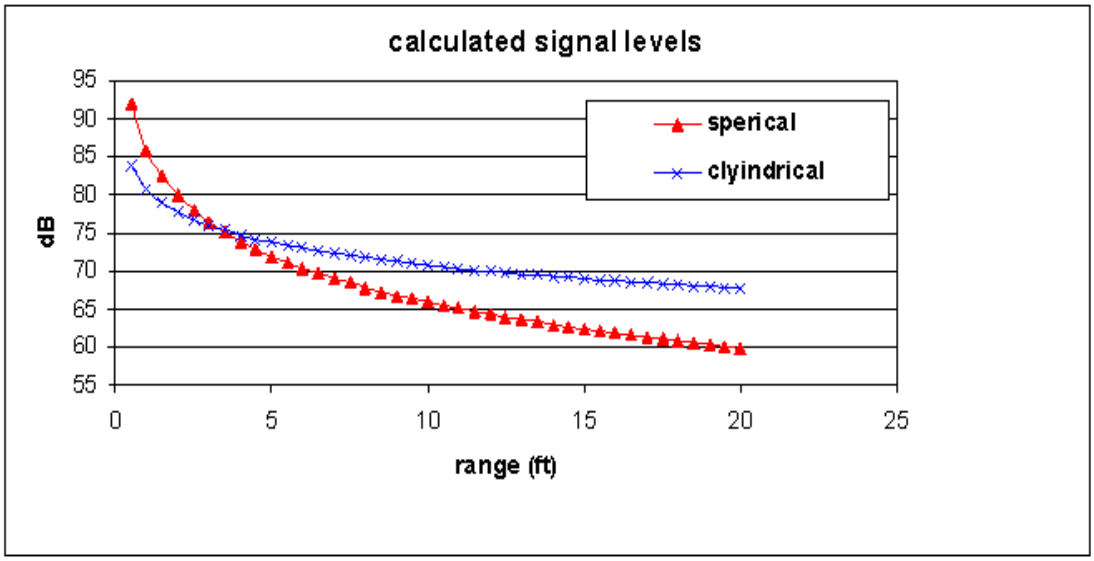

Figure 5.1 Intensity of the sound drops as range from the source increases. If sound were behaving in accordance with the cylindrical spreading model, amplitude differences at the microphones beyond 10 feet would be hard to detect. If the random controller guides the robot to this range or beyond, any information recorded during this time will not be usable.

The 3rd controller, dubbed 'purposeful but random', used a goal directed/feedback approach to collecting a sufficient memory set. The controller used sensor feedback to keep doing actions that would move it closer to its goal state. As long the feedback indicated that things were going well it remained doing what it was doing, but as soon as this was not true, it randomly chose another action. Since the selection of the new action is random, examples of correct actions are distributed relatively evenly. That is, there will be about an equal amount of examples in which a right turn was the correct action as there will be for going straight and left. The outline of the algorithm for a following robot is shown below.

The variables are defined as follows:

L1: left sensor value.
L0 : previous left sensor value.
R1 : right sensor value.
R0 : previous right sensor value.
currentD : current action (-1 for left, 0 for straight, 1 for right)
newD : new direction that the algorithm sends back

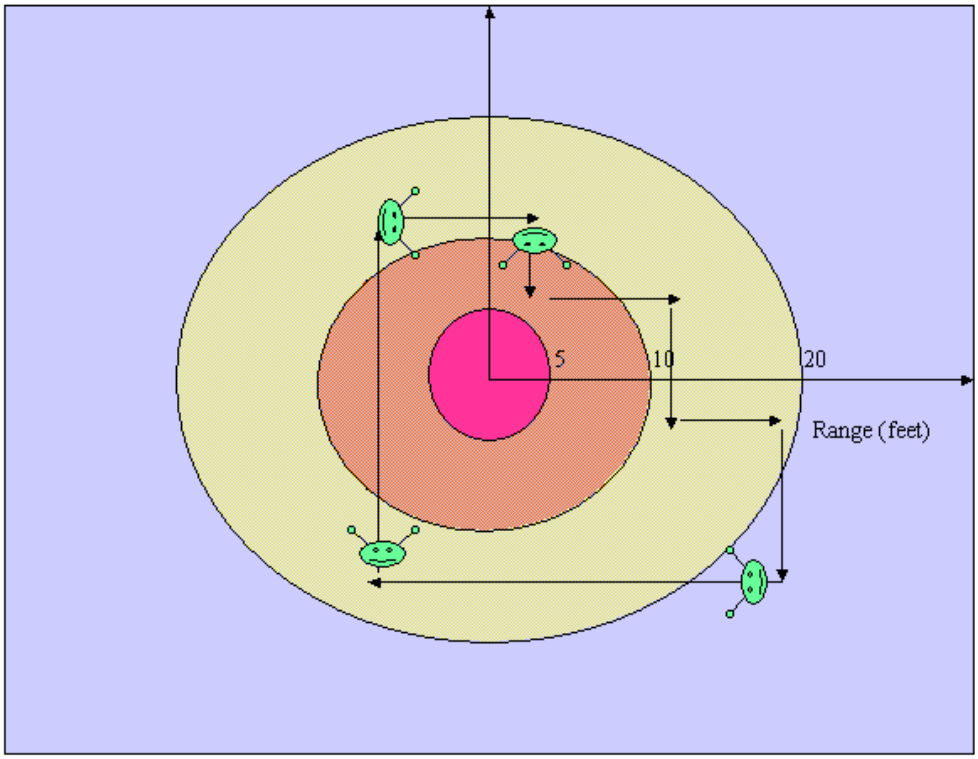

Figure 5.2. A pattern that was used to collect a recent memory. The arrowed line is the robot's track. The green faces with antenna denote the robot and its orientation to the source as it traverses its pattern. The source is in the middle of the plot. The colors going from bright to cool indicate the amount of information available with respect to the distance from the source. As can be seen from the plot, most of the track was in areas where there was not a lot of information available.

The algorithm is shown below:

```
/* Find larger of two sensor values. */
if (L1 > R1) {
        big = L1;
        little = R1;
}
else {
        big = R1;
        little = L1;
}

/* If both sensors are gaining in intensity and close to each other, go to a
neutral state, that is go straight. */
if (((Ll > L0) AND (R1 > R0)) AND (little * 1.05 >= big )){
```

```
        newD = straight;
}
else {
        /* If one sensor is gaining, keep doing what is currently being
        done. */
        if ((R1 > R0) OR (L1 > L0)){
                newD = currentD;
        }
        /* Otherwise try something new. */
        else {
                newD = random choice of left, straight, or right
        }
}

/* Update previous left (l0), right (r0) values and record the current
direction. */
L0=L1;
R0=R1;
currentD = newD;

return newD;
```

The algorithm works by allowing the robot to turn in the direction in which the signal strength at its sensors is increasing until they are almost the same signal strength, then it stops turning. If the robots sensors are not gaining signal strength, it makes random changes to its direction until it finds a heading that causes the signal strength to go up. Once the signal strength is going up, it remains doing whatever random change it selected, until both sensors are showing increases in signal strength and they are about equal. Figure 5.3 illustrates the affect of the algorithm on the robots path. Notice how much more of the path is in the areas where usable information can be collected. Not only is more of the path in information rich areas, but by looking at the shape of the path it can be seen that there are more examples of left, right, and straight actions that take the robot towards the sound source.

The algorithm is purposeful in that it knows what it 'likes' i.e. it is based on the programmed goal of the system: increasing signal strength at both sensors. It is random because it does not associate its sensor readings with its actions, that is when goals are not being met, it picks a random action. To summarize, the effect of the algorithm is to keep the robot moving around the signal source, much like a moth flies around a bright street light at night, providing many positive examples from which learning can occur. The next section discusses reactive learning techniques using recent memories collected using operators and the techniques discussed in this section.

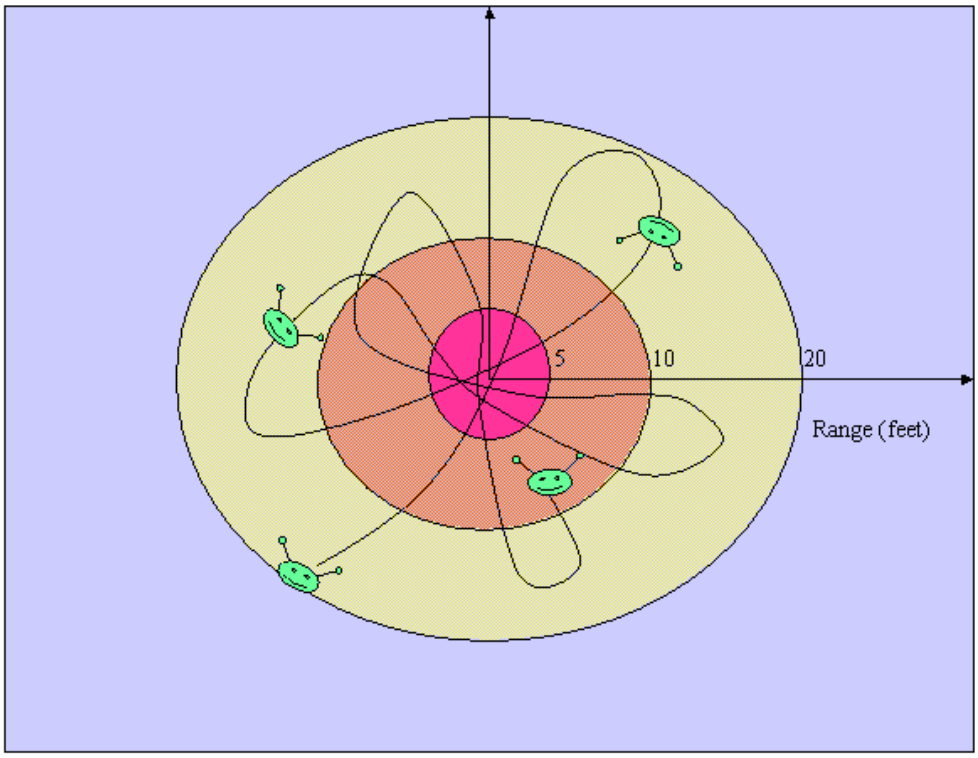

Figure 5.3. The path that the 'purposeful but random' controller generates. As in figure 5.2, the green smiley faces denote the robot and its orientation as it follows the path. The spiral looking curve is the robots path. As in figure 5.2 the colors indicate the amount of usable information with respect to range.

5.3 Reactive Learning Using Sensor/Action Pairs

The reactive learning algorithm uses a recent memory, consisting of a sequence of sensor/action pairs, to learn which actions have a direct affect on the pursuit of goals. This algorithm is concerned primarily with single sensor/action pairs. Direct means that an action taken at time step T (0) influences sensor readings in time step T (1) in a measurable way. The recent memory is searched for these occurrences. If the action had a positive effect it is put into a positive example set, and vice versa. Using these sets, a GA is used to train a neural network to estimate the correct actions based on the sensor values. Figure 5.4 below outlines the reactive learning process.

In the diagram shown in figure 5.4, the recent memory is sent to the intensity filter. The intensity filter finds all occurrences in which the sensor readings at time T (1) strongly indicate that a correct action was taken at time T (0). For the case of robots learning to follow, the intensity of the left and right sensor readings at T (1) would be higher than those at T (0). The amount of improvement in the sensor strengths is currently an operator settable parameter, usually set to 20 to 30 percent improvement between T(0) and T(1). All occurrences of the just described situation are put into the positive example set, and examples in which the opposite occurred, i.e. sensor readings at T (1) that are substantially weaker than those at T (0) are put into the negative example set. Neutral

examples are not used because the sensor readings at time T (1) do not indicate whether correct actions or incorrect actions were taken at time T (0).

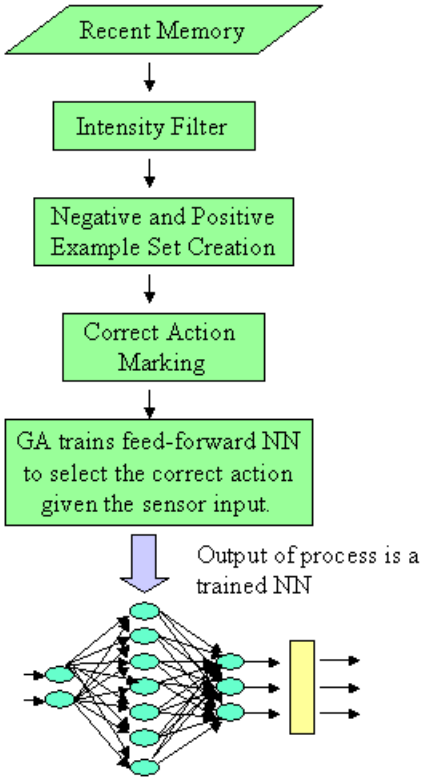

Figure 5.4. The reactive learning process. Key to the process is the intensity filter. It makes sure that the memory is composed of examples in which actions at time T (0) had a direct influence on sensor values at T (1). As a part of the robot architecture, this process would be called when an observer function notices goals are not being met. This condition could occur for a variety of reasons, including changing environmental conditions, or fouled sensors.

Once the two sets are created, the actions are marked. For each example in the positive example set, it is assumed that the action taken at T (0) caused the improvement noticed in T (1), so the action at T(0) is marked as correct. In the negative example set, it is assumed that the action at taken at T (0) caused the deficit in the goal parameter noticed at T (1), so the action taken at T (0) is marked as incorrect. Since the correct action is not known for the negative examples, the training file is set up to request the GA to reward the selection of an alternate action from the remaining actions available.

Once all the actions have been marked, the training file is written. The GA uses the training file to train a FFNN in a similar manner as to that discussed in the previous chapter. The inputs to the network are the sensor readings. The output is the action to be taken. The fitness function rewards networks that choose the action associated with those sensor values. The GA creates each successive generation of networks from the successful networks of the current generation. The stopping criteria for the GA has been implemented in two ways. In the first, discussed in chapter 4, the GA executes a fixed

number of generations. Since the robots onboard computers do not have as much processing power as the development computers, a provision to stop the GA when there has been no improvement for a specified number of generations was implemented. The best scoring neural network of the last generation is used as the reactive controller.

The steps below outline the learning from a recent memory process for a robot whose goal is to follow its leader in a formation maneuvering task. Note that only the positive example set is used in this outline.

1. Let a follower robot record a memory consisting sensor/action pairs while seeking to either a fixed or moving sound source. The robot can be guided by an operator, or by some other algorithm, such as the classic logic method, another neural network, or the random but purposeful controller.
    a. A sensor action pair SA consists of a left microphone value, L, a right microphone value, R, and an action A.
    SA = {L, R, A}
    The L value of a sensor action pair SA is indicated by $SA \rightarrow L$
    The R value of a sensor action pair SA is indicated by $SA \rightarrow R$
    The A value of a sensor action pair SA is indicated by $SA \rightarrow A$
    b. A recent memory M is defined as a sequence starting at time 0 and completing at time n-1 of n sensor/action pairs.
    M = SA [0] …..SA [n-1]
2. Create training sets of positive examples, P, and negative examples, N, from M using the following logic:

```
/* Initialize P and N to the empty set. */
P = N = ∅ ;
For j = 0 until j = n -2
      /* total sound energy = left mic value + right mic value */
      /* E0 = total sound energy at time 0 */
      E0 = SA[j]→R + SA[j]→L;

      /* E1 = total sound energy at time 1 */
      E1 = SA[j+1]→R + SA[j+1]→L;

      If (E(1) >E(0)) then
            /* Sensor/action pair at time 0 is put in positive example set. */
            P = SA[j] ∪ P;
      Else
            /* Sensor/action pair at time 0 is put in negative example set. */
            N = SA[j]  ∪ N;
End for
```

3. Using the set of positive examples as the new training set, let the GA based technique generate a new neural network. In order to increase efficiency of the GA, the current best neural network can be used to seed the initial population.

Step 2 implements the local optimization that is key to reactive learning. As stated earlier, for this step to be valid, the nature of the signal that the sensors are sampling has to be such that the results of an action taken at a particular time step can be sensed during the next time step. The intensity filter (figure 5.4) helps to eliminate cases where the effects of low amplitude and/or noise hinder action/sensor coupling.

As previously stated, in some cases the negative example set can also be used. To do this, the fitness function in the GA is programmed to encourage selection of an alternate action. The reasoning is that if a given that the action does not yield positive results for the next time period, one of the other choices would have been better. A training set that is sufficiently representative of the conditions that will be encountered will have enough examples to resolve any ambiguities.

Table 5.5. Data collected from a typical memory. Note that the action taken at T(1) seems correct, but would be put into the set of negative examples.

Time T(n)	Left Sensor Value	Right Sensor Value	Total energy	Action Taken (left, straight, right)	Example Set
0	10	0	10	Left	positive
1	5	6	11	Straight	negative
2	5	5	10	Straight	positive
3	7	7	14	Straight	

That being said however, for these techniques to work choices made at time T (0) need to be accurately reflected in the sensor values by time T (1) more often than not. If this is not true, the training set will be too ambiguous to encourage correct decisions. Looking at table 5.5 above, it can be seen that the total energy at T (0) is 10 (left sensor value + right sensor value). At T (1) the total energy is 11. According to rule 2 from the learning from a memory process, T (0) would be put into the positive example set. Using this same rule T (1) would be labeled a as a negative example, although it is clear from the data at T (2) and T(3) that straight is the correct choice at time T(1). Using negative examples, the GA would erroneously encourage an alternate choice, left or right. With a limited number of occurrences of these erroneous examples, the GA can form a working neural net, but as their numbers increase, the data set becomes more ambiguous, causing the resulting network to be less effective.

This algorithm was used to create controllers that were tested both in simulation and using the mobile robots. Memories collected in the simulator worked well when using both the positive and negative example sets, but memories collected using the mobile robots worked only when using the positive example set. The unpredictable nature of sound in the lab populated the training set with negative examples similar to the one illustrated in table 5.5, rendering the learning process ineffective.

Ah-Hwee Tan et al.[51] compared a reactive plan execution system to a reactive learning algorithm in a simulated mine avoidance application. In this system the autonomous agent used a small array of sonars to detect the mines as it navigated towards its goal location. The plan execution system was a rule based system relying on a-priori knowledge while the learning system used a reinforcement based algorithm. Both systems made heavy use of data quantization in order to reduce the state space. To train the learning system, 1000 trial runs consisting of 30 sense-act-learn cycles were used. Using this method the learning system returned performance similar to that of the plan execution system. The reactive learning algorithm discussed in this section differs in that it avoids the large number of trial runs during the learning process. While this is not as important when running in a simulation, it has implications when onboard a physical robot because it saves time, battery power, etc.

The next section discusses a technique that was used to generate extra examples from those already collected. This process creates new memories to form a more complete set of experiences. The section briefly describes the concepts and presents the algorithm.

5.4 Sensor Mirroring

As mentioned earlier, sometimes it is preferable to use only the positive example set in training. The key to doing this is to make sure that the positive example set is as complete as possible. In this context, complete means that there is a positive example for each situation that is likely to be encountered during the robots operation. A 'mirroring' process was developed to help ensure completeness of the experience set. The mirror process makes the assumption that in the system there is symmetry in the right and left sensors and the actions that are associated with values coming from these sensors. While this approach is not necessarily applicable to all learning situations, it is certainly valid for a large set of applications including the formation maneuvering problem. For example, if a strong signal on the left sensor indicates that the robot should turn left, then the converse is also true; a strong signal on the right sensor should induce a turn to the right. If a memory contains more of one kind of example than the other, by using symmetry the memory can be made to contain an equal amount of both. The algorithm below describes the mirroring process.

```
/* Mirror process for reactive memory preparation. */
For each positive example in P
        temp = left sensor value;
        New left sensor value = right sensor value;
        New right sensor value = temp;

        If (action = left) then {
                New action = right;
        }

        If (action = right) then {
                New action = left;
```

```
        }

        else {
                New action = straight;
        }
End for
```

Sensor mirroring worked well for completing recent memories in which certain sensor/action sequences were under-represented. While it is a useful tool, it relies on the existence of symmetry that can be a-priori identified and designed into the system by the human creator. It was used heavily during the first part of this research, but the development of the random but purposeful controller and more effective fitness functions lessened the need for the sensor mirroring process.

5.5 Non-Reactive Learning Using Runs of Sensor/Action Pairs

So far all the neural networks implemented have had two inputs, the current reading for the left and right microphones. These networks have been shown to be useful in guiding both simulated and mobile robots in formation maneuvering tasks. But because of their reactive nature the resulting motion of the robots tends to be somewhat abrupt and because it does not consider past values, reactionary control cannot differentiate between heading directly towards the signal source and directly away from it. In order to solve this problem examination of past values is necessary. By doing this the robot can determine if its current state is better or worse than its previous one, a fundamental shift in the approach taken to goal accomplishment in the previous work.

The behavior routine effectively accomplished this in simulation by keeping a running history of the signal strength. It arbitrated its behaviors based on this running history. This strategy proved to be inadequate in the lab because of the fickle nature of the lab acoustics, but given more development the strategy may have been effective. Because it did not lend itself to being machine produced this algorithm was not pursued further. Never the less, it did show that considering past values is key to solving the inadequacies of the reactionary approach. What is needed is a machine producible neural network that considers past values.

As stated earlier, both FFNN and recurrent neural networks can consider past values. A recurrent neural network is more flexible than a FFNN because it can consider an arbitrary number of past values. It does this by having cycles[52] (closed loops) in its flow structure; see figure 5.6. The effect of past values is regulated by the complexity of the structure, coupled with the weights on the connections. This structure can be much more complex than that of a FFNN, making the solution space much larger and increasing the possibility of getting caught in local minima. A FFNN can only consider a fixed number of past values, but because it does not have cycles, its structure is straightforward.

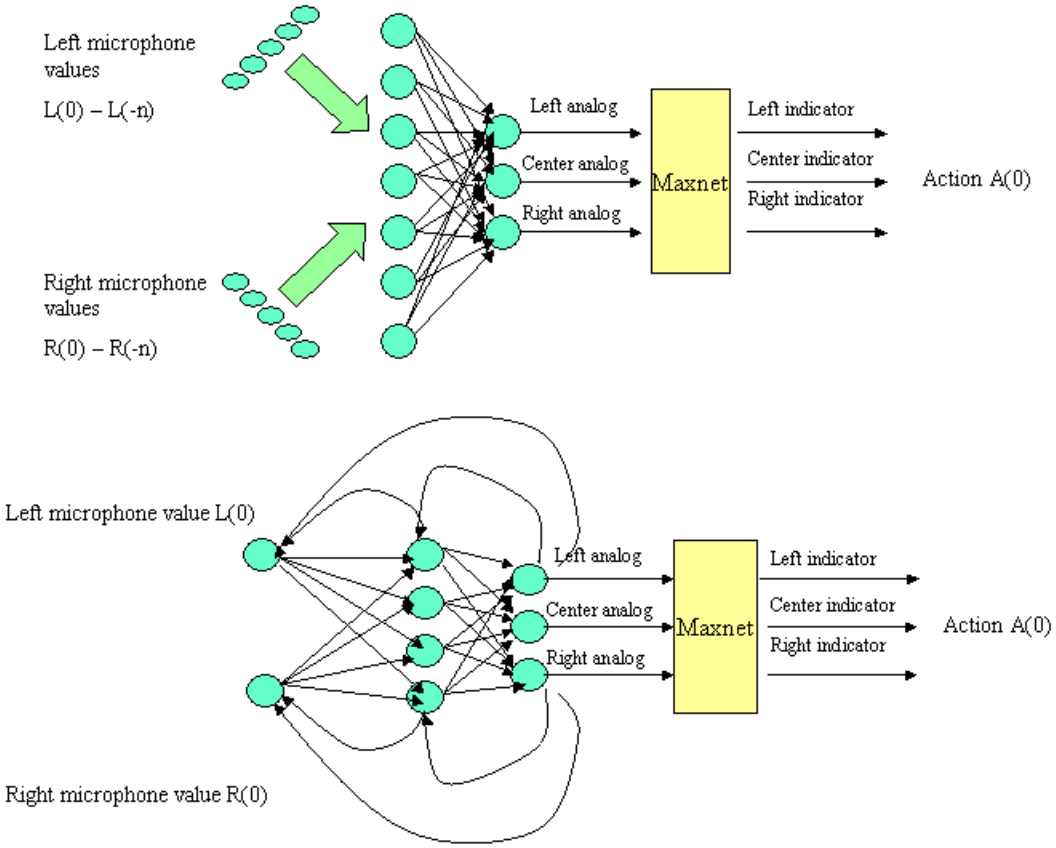

Figure 5.6. A FFNN with past inputs in the top figure and a recurrent network in the bottom figure.

The network at the top of the figure is a FFNN with 5 inputs from each microphone. The inputs consist of L (0) and R (0), the current microphone values, and L (-1) to L (-4) paired with R (-1) to R (-4), the four most recent past microphone values. The recurrent network shown at the bottom of the figure has only the current microphone values as inputs. The cycles are made by the arrows that loop back from the output layer to the hidden layer and input layer, and the hidden layer to the input layer. In the interest of simplicity the FFNN with multiple inputs was selected to handle the non-reactive tasks, with the provision that if the recurrent network was needed, it would be implemented.

Because the non-reactive system is considering past actions, its job is to produce actions that directly or indirectly optimize goals, as opposed to the reactive system, which attempts to directly optimize goals in each time step. This is an important distinction because not all problems are solvable by a series of actions that get closer to the solution at each time step. A simple example of this is an obstacle directly in the path between a robot and a goal point. The robot must make movements that are not directly toward its goal to reach its goal. Thus, the non-reactive system is capable of solving a larger class of problems than the reactive.

Considering the case of a robot heading directly away from its source, the reactive system has no answer. Because the non-reactive system can look at past values it can determine

that over time the source strength is diminishing and take an appropriate action, such as turn. But, because the non-reactive system works with sequences of events, the reactive FFNN training methods presented thus far will not suffice. The next section describes the methods developed to train non-reactive FFNNs.

5.5.1 Training Non-Reactive FFNNs Using the Principle Parts Method

One of the main requirements for training non-reactive FFNNs is that the inputs be contiguous in time. This requirement does not exist for reactive networks. Recall from the previous discussions that the main requirement for selecting good examples is that the signal strength at the next time step is greater than the current signal strength. If this condition were true it was assumed that the action taken at the current time step was responsible for the gain in signal strength. In that strategy sensor/action pairs are examined, and single entries are selected and entered into the training set. For the non-reactive FFNNs time and the ordering of events needs to be preserved, thus a different methodology is used to create the training set, but the same basic GA is used to find the weights for the FFNN. This section describes the techniques used to create the training sets.

In order to create a good training set from a recent memory, the memory must have a wide variety of examples from which to learn. Given that this is true, and assuming the goal is expressed in terms of sensor values, the memory can be separated into different segments depending on how the sensor values are reflecting the goal. As before, for the target application in this work, the goal was to maximize the energy received at the left and right microphones. Expressing the goal as a time series it was viewed as sequence of four basic parts. They were goal parameter rising, dropping, remaining stabile, and fluctuating. Figure 5.7 below shows an illustration of a time series and the four principle parts used to construct it.

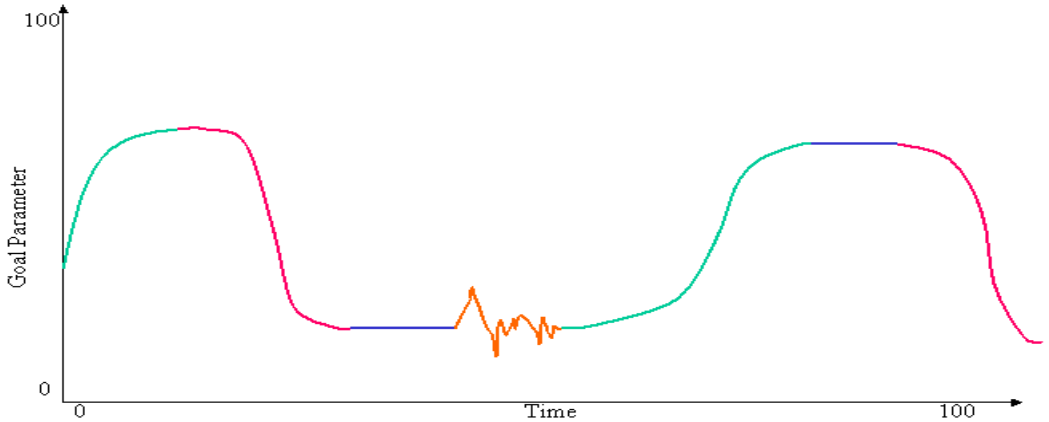

Figure 5.7 A time series of the goal parameter. The line is drawn in different colors to illustrate the principle parts that it is made of. The green part of the line fits in the category of rising parts, the magenta part in the dropping category, the blue part in the remaining stabile category, and the orange part in the fluctuating category.

Each principle part was further split into segments in which the action was constant. So in the robot following application, the recent memory was broken down into a sequence of the four principle parts. Then to create the positive example set, each goal rising segment from the recent memory was split into sub-segments, referred to as runs, in which the actions were constant. Typically the positive example set would contain 40 to 50 examples (the sample rate is typically 20Hz), each about 35 time slices in length. The distribution of robot going straight and sensor readings increasing, robot turning left and sensor readings increasing, and robot turning right and sensor readings increasing, were somewhat equal, as a result of using the random but purposeful controller. The logic below explains the breakdown of the memory.

1. Define the Recent Memory as follows:
    a. SA : sensor/action pair consisting of the ordered 4 tuple {L, I, R, A}whose members are defined below.

        L: left relative microphone value
        I : sound intensity value
        R : right relative microphone value
        A : action; The action can be left, straight, or right

        The relative values L and R are found using the algorithm detailed in section 4.3.

        The intensity value, I, is the average of the left and right raw intensities.
            leftRaw : the value the sound system returns for the left microphone
            rightRaw : the value the sound system returns for the right microphone
            I = (left raw + rightraw)/2

        The L value of a sensor action pair SA is indicated by $SA \rightarrow L$
        The I value of a sensor action pair SA is indicated by $SA \rightarrow I$
        The R value of a sensor action pair SA is indicated by $SA \rightarrow R$
    The A value of a sensor action pair SA is indicated by $SA \rightarrow A$

    b. A recent memory M of length n is defined as a sequence starting at time 0 and completing at time n-1 of sensor/action pairs.
        M = SA [0] …..SA [n-1]
2. Define the principle part categories RC, SC, DC, and FC as follows:
    a. RC : rising category. In the rising category the intensity value is constantly rising. The rising category is further subdivided into subsets in which the action is consistent.
        RC_left is the rising category in which all actions are left turns.
        RC_straight is the rising category in which all actions are straight.
        RC_right is the rising category in which all actions are right turns.

        $SA[j] \in RC\_left$ if $SA[j] \rightarrow I > SA[j-1] \rightarrow I$ and $SA[j] \rightarrow A$ = left
        $SA[j] \in RC\_straight$ if $SA[j] \rightarrow I > SA[j-1] \rightarrow I$ and $SA[j] \rightarrow A$ = straight

$SA[j] \in RC\_right$ if $SA[j] \rightarrow I > SA[j-1] \rightarrow I$ and $SA[j] \rightarrow A = right$

$RC = RC\_left \cup RC\_straight \cup RC\_right$

b. SC : stabile category. In the stabile category the intensity value is not changing. The stabile category is further subdivided into subsets in which the action is consistent.
SC_left is the rising category in which all actions are left turns.
SC_straight is the rising category in which all actions are straight.
SC_right is the rising category in which all actions are right turns.

$SA[j] \in SC\_left$ if $SA[j] \rightarrow I = SA[j-1] \rightarrow I$ and $SA[j] \rightarrow A = left$
$SA[j] \in SC\_straight$ if $SA[j] \rightarrow I = SA[j-1] \rightarrow I$ and $SA[j] \rightarrow A = straight$
$SA[j] \in SC\_right$ if $SA[j] \rightarrow I = SA[j-1] \rightarrow I$ and $SA[j] \rightarrow A = right$

$SC = SC\_left \cup SC\_straight \cup SC\_right$

c. DC : dropping category. In the dropping category the intensity value is constantly dropping. The dropping category is further subdivided into subsets in which the action is consistent.
DC_left is the dropping category in which all actions are left turns.
DC_straight is the dropping category in which all actions are straight.
DC_right is the dropping category in which all actions are right turns.

$SA[j] \in DC\_left$ if $SA[j] \rightarrow I < SA[j-1] \rightarrow I$ and $SA[j] \rightarrow A = left$
$SA[j] \in DC\_straight$ if $SA[j] \rightarrow I < SA[j-1] \rightarrow I$ and $SA[j] \rightarrow A = straight$
$SA[j] \in DC\_right$ if $SA[j] \rightarrow I < SA[j-1] \rightarrow I$ and $SA[j] \rightarrow A = right$

$DC = DC\_left \cup DC\_straight \cup DC\_right$

d. FC : fluctuating. In the fluctuating category the intensity value is constantly changing. SA sequences that do not fit in any of the RC, SC, or DC categories are put into the FC category by default.

3. A run is defined as contiguous sequence in time of SA pairs from any one of the sets RC_left, RC_straight, RC_right, SC_left, SC_straight, SC_left, DC_left, DC_straight, or DC_right. A looping process that evaluates the recent memory M by examining consecutive SA pairs creates the runs. Once the subcategory within the RC, SC, or DC set of an SA (SA[m], SA[m+1]) pair is determined the process checks to see if the next pair (SA[m+1], SA[m+2]) falls in the same category. The process continues until an SA pair does not fall within the current subcategory. If the run length is beyond a certain minimum, usually 16 SA pairs, the run is used as a training example. This process is continued until all runs within the memory M are identified and turning into training examples. Selection of training actions for the runs are described later in the text.
4. Using this nomenclature the recent memory M is comprised of the 4 principle parts.

$$M = RC \cup SC \cup DC \cup FC$$

Figure 5.8 below shows a plot in which the robot's path is separated into runs in which the action is constant and the intensity parameter is either rising, dropping, remaining stabile, or fluctuating. The different colors indicate which principle part set the segments fall into. As in figure 5.7, green segments mean that the robots sensors are reporting an energy gain, blue segments correspond to a stabile energy situation, red indicates an energy loss, and orange indicates fluctuations. Looking at the diagram it can be seen that the robot is heading towards the sound source in run denoted by line segment 0 –1 and heading away in the run denoted by segment 1 to 2. It begins losing energy in run 2 –3, and in run 3 – 4 the random but purposeful controller is searching for a good direction, hence the fluctuations in the path. In the run denoted by line segment 4 – 5 the controller randomly chose to turn left and since there was energy gain, it kept going left, and so on. Table 5.9 describes the runs and what categories of principle parts they fall into.

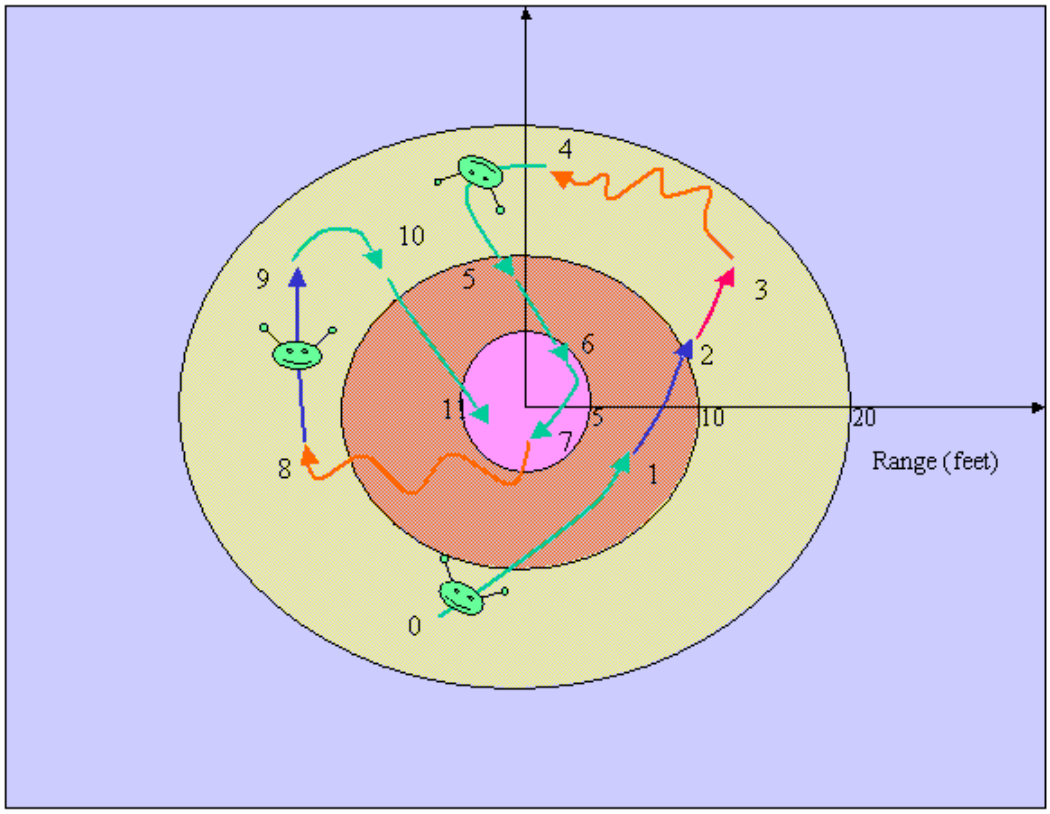

Figure 5.8. Part of a path in a recent memory. As in figure 5.7 green indicates energy gain, red indicates energy loss, blue indicates a stabile energy situation, and orange indicates a fluctuating energy situation.

For examples created from the runs in the rising category, the fitness function in the GA encouraged the FFNN to select whatever action had been done in that run. As in the positive examples in the reactive training method discussed earlier, since the goal parameter was rising it was assumed that the action was correct. While this assumption is not always correct, it is correct most of time in a recent memory generated by the random

but purposeful controller. The example set in which the goal parameter remained stabile was treated the same as the positive example set, but the set in which the goal parameter drops was treated differently.

Table 5.9. This table organizes the segments shown in figure 5.8 into the principle parts categories that they would fall into.

Start of Run	End of Run	Action	Principle Part Category
0	1	Straight	Rising
1	2	Straight	Stabile
2	3	Straight	Dropping
3	4	Random Combination	Fluctuating
4	5	Left turn	Rising
5	6	Straight	Rising
6	7	Right Turn	Rising
7	8	Random Combination	Fluctuating
8	9	Straight	Stabile
9	10	Right Turn	Rising
10	11	Straight	Rising

When the goal parameter is dropping the correct action is unknown. To determine the action to be used for a run that falls into the dropping category, an adjacent or overlapping rising run must be found. Once the overlapping example is found, its action is used as the correct action for the dropping run. It is used because it is the first correct action that occurs since the negative energy gain. If there is no adjacent or overlapping rising run, the dropping run is not used. Fluctuating runs are not used to create training examples at this time because there is no programmatic method of determining what the correct action is for these states. It is this technique that is used to solve the forward/backward ambiguity problem. Table 5.10 below illustrates this concept.

Approaching the forward/backward ambiguity problem using this method is remarkably different than the method described by Shah[31] . Both make use of data collected over time, but where Shah's method is rule based, the approach described here is learned. The implication is that as long the environment and the rules remain in sync, performance of Shah's method should be good, but given changes to the environment, a well refined learning technique in a robot control architecture in which it can be utilized should be more flexible.

Once the runs were so categorized, training examples were created and the non-reactive FFNN could be trained. Typically the network would look at the six most recent inputs from each sensor. The GA was used as before to find the weights and the most-fit FFNN of the last generation was used to control the simulated robot.

The fitness function used in the GA with the principle parts training set is based on the percentage of correct actions produced by the FFNN for the training set. It is based on the percentage instead of the total number so that one type of action does not overshadow

another. To illustrate, if a recent memory contained 60 runs, consisting of 25 left turns, 25 straights, and 10 right turns, a FFNN could score a 50 out 60 and not have to be able to recognize when it needed to turn to the right. By using the percentage method, the same FFNN would score a 2 out of 3, because it got 25/25 left turns correct, 25/25 straights correct, but 0/10 right turns correct. By using this learning method each action is equally important, reducing the need to use sensor mirroring to construct a complete experience set.

Table 5.10. This table shows a dropping run in which the action is straight that is adjacent in time to a rising run in which the action is left. Looking at the memory index column, it can be seen that the DC_straight run ends at time 5, and the RC_left run begins at the next time slot, time 6. To use the dropping run in training, the left action from the rising run is assigned to it because it is assumed that the left action is what caused the intensity value to begin rising.

Memory index	Intensity value	Is a member of set	Action
0	6	DC_straight	straight
1	5	DC_straight	straight
2	4	DC_straight	straight
3	3	DC_straight	straight
4	2	DC_straight	straight
5	1	DC_straight	straight
6	2	RC_left	left
7	3	RC_left	left
8	4	RC_left	left
9	5	RC_left	left
10	6	RC_left	left

The next section describes results using the reactive and non-reactive FFNNs that were created using techniques described in this chapter. It also describes testing undertaken to determine if the non-reactive FFNN could solve the forward/backward ambiguity problem.

5.6 Experimental Results

Tests in the both the simulator and in the lab were carried out with the reactive neural networks. As stated in chapter 4, since the infrastructure for precise position measurements is still being developed the Lab tests consisted of a formation maneuvering task in which the follower robots follow a leader in either a series of laps around the lab or a follower follows a leader making random course changes. The reactive FFNNs were able to control a robot in these tests. Figure 4.15 from the previous chapter is representative of the lab tests.

Simulator tests were also conducted. The simulator tests measured the leader to follower robot's distance and the heading difference between the leader and follower robots' courses. In the test the lead robot traversed two laps around a preprogrammed course in the simulator. The tests are started in formation, by maneuvering the leader follower pair

to the starting position and then engaging the automatic navigation control for the leader robot. The robot's turning rates and speeds were set the same for all tests. Figure 5.11 illustrates the test track in the simulator.

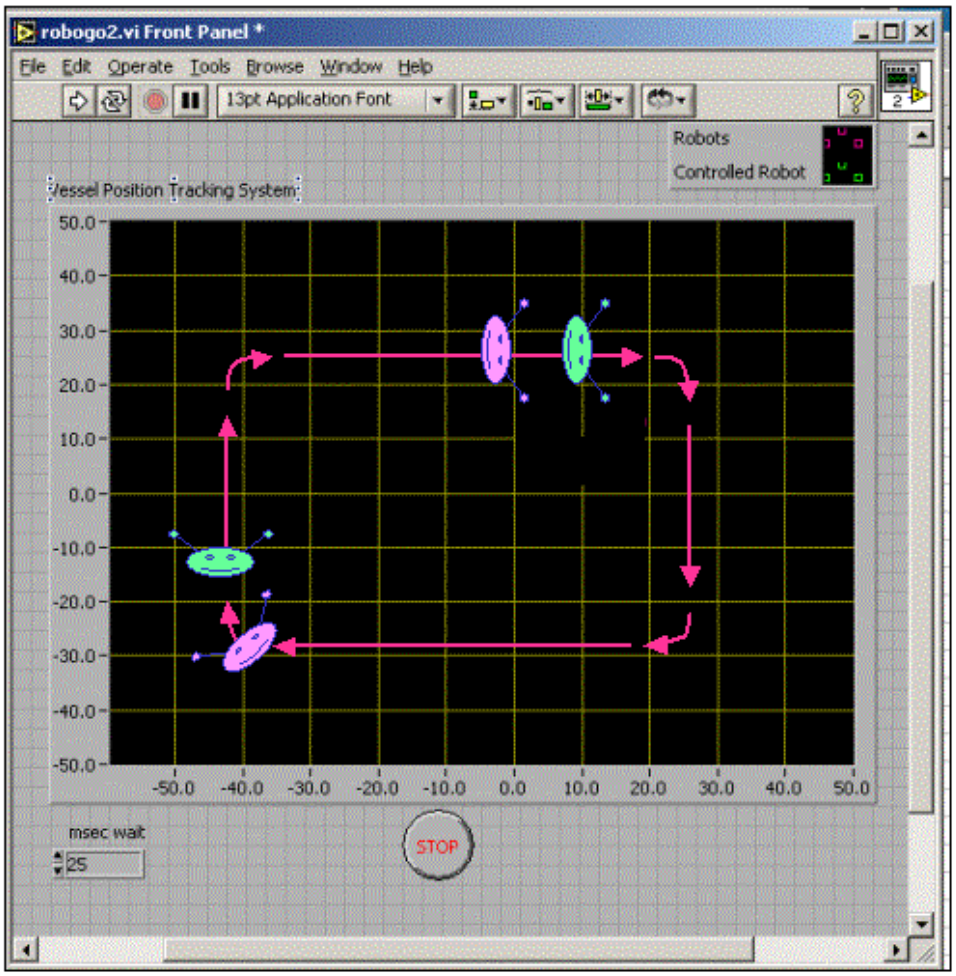

Figure 5.11. The test course for the following robots. The lead robot (green smiley face) is controlled by the simulator. The follower robot (pink smiley face) is controlled by the current controller being tested. Over two laps the inter robot distance and course differences are recorded.

The tests are started in the upper left hand corner of the track, when the leader/follower combination is in formation and tracking (see figure 5.11). The test is complete when the combination rounds the upper left corner at the end of the second lap and is heading towards the corner to the upper right.

Figure 5.12 shows the average distance between the leader and a follower controlled by a typical reactive neural network trained as discussed earlier in section 5.3. The distance between the two remains stabile except during the turns. The turns are the 8 places in the curve where the distance fluctuates.

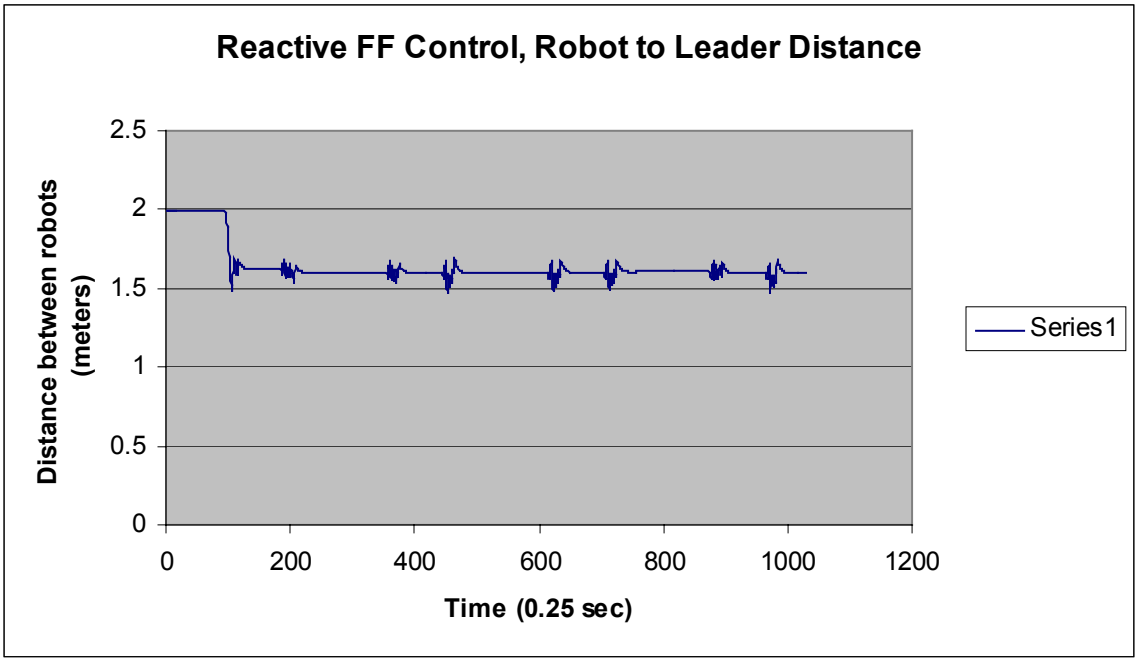

Figure 5.12. The leader/follower distance over two laps. The follower robot is controlled by a reactive FFNN. As can be seen in the figure, the robot closed the distance between it and its leader and maintained it throughout the run. The fluctuations in the line are when the leader is making a turn on its preprogrammed course. As can be seen there are 8 turns over two laps.

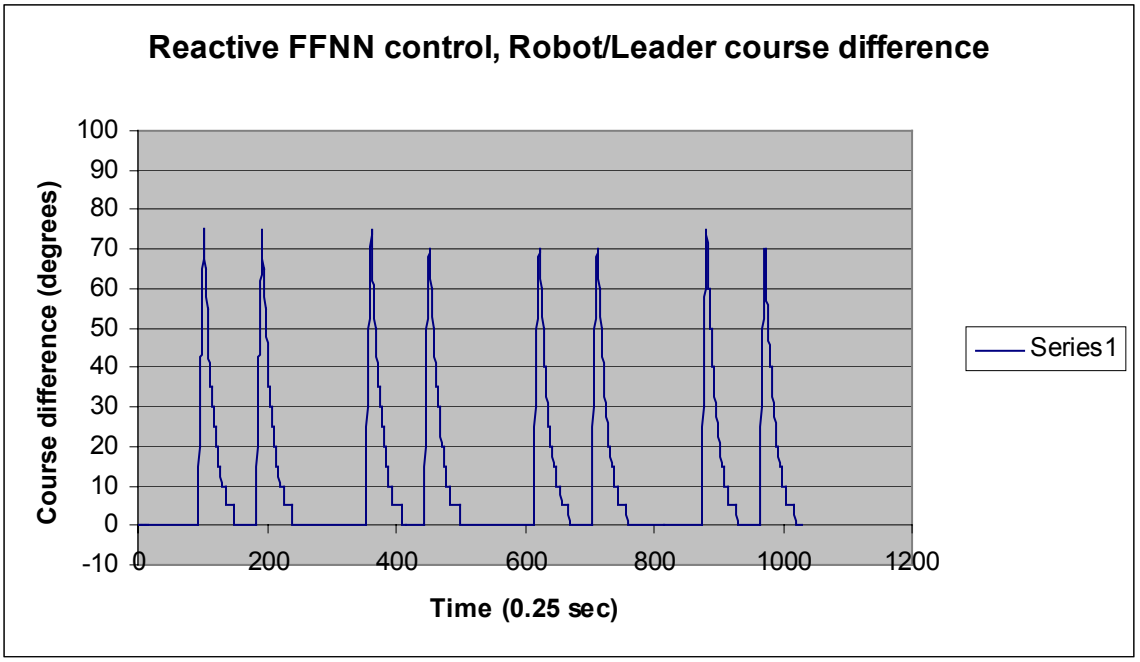

Figure 5.13. The heading difference between the leader and the follower robot controlled by a reactive FFNN during a two lap test in the simulator. Notice that each of the 8 peaks (the peak denotes a course change by the leader) occur at about the same time as the fluctuations occur in the curve in figure 5.12.

Figure 5.13 shows the heading differences measured during the same test. The graph has 8 peaks in it, each one corresponding to the leader making a course change. For comparison 5.14 and 5.15 show a similar test run using the classic logic controller. Recall from chapter 4, it is a reactive controller based on Braitenberg's robots.

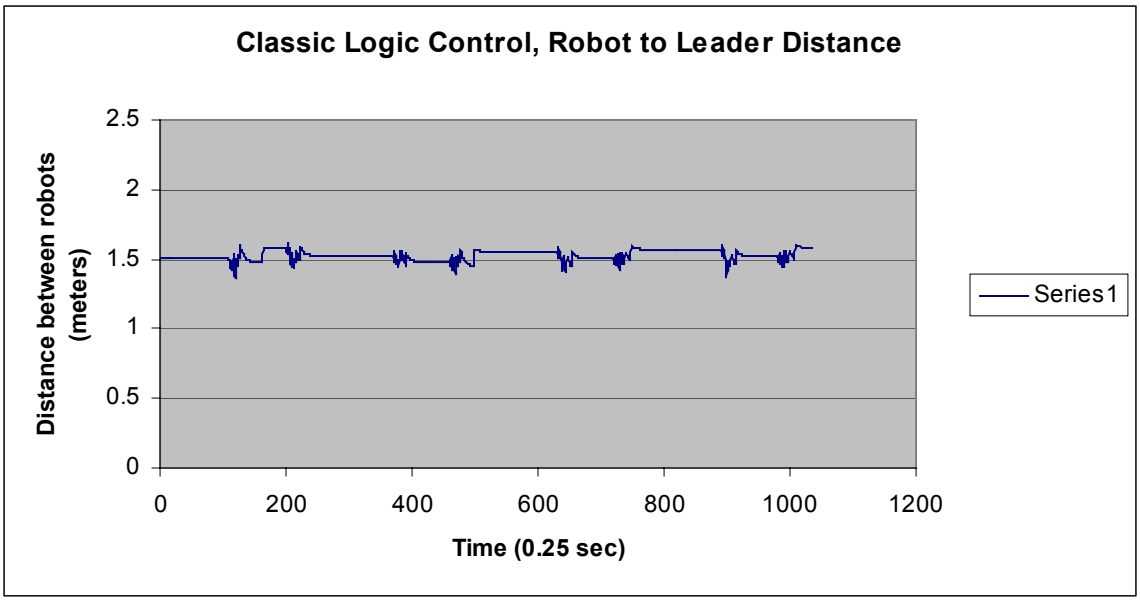

Figure 5.14 The leader/follower distance for the two lap test when the follower is using the classic logic controller.

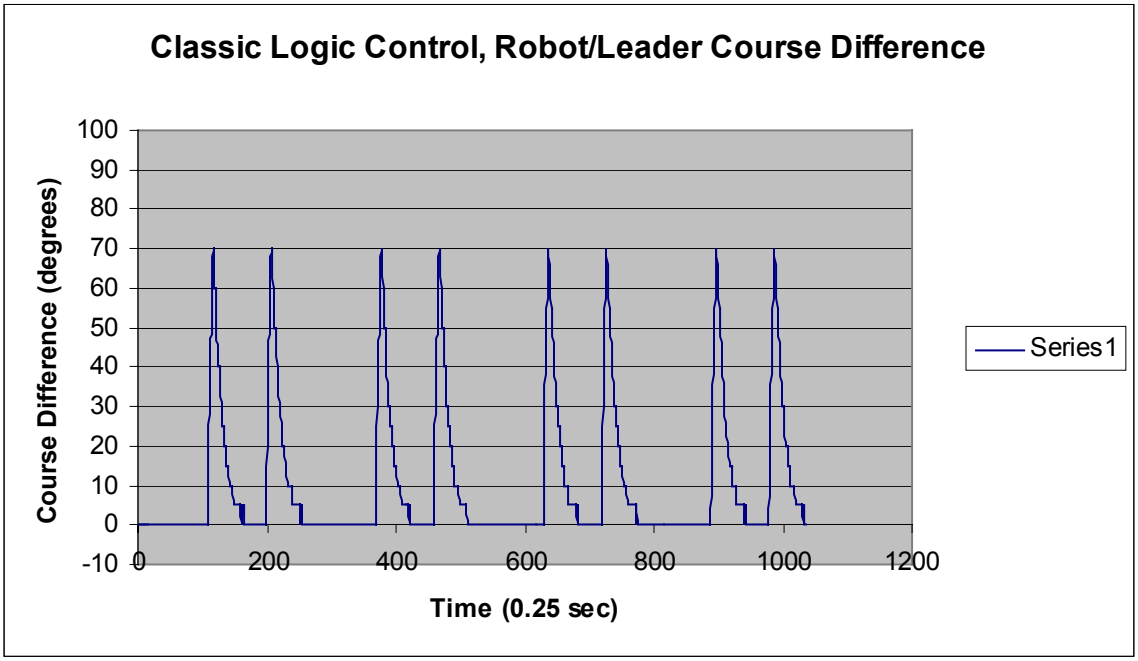

Figure 5.15. This figure shows the course difference between the leader and follower when the follower is using the classic logic controller. Each peak corresponds to when the leader is making a turn.

Comparing figures 5.12 and 5.14, it can be seen that the inter-robot distances are very similar, but the curve shapes are quite different. The curve in 5.12 is much smoother overall, indicating that the distance between the robots varies less with this controller. The reason for this is that the classic logic controller tends to oscillate when it is nearly centered. The straight percentage parameter damps this phenomenon but has the adverse effect that it causes the robot to take more time to match its course with the leaders. These oscillations can be seen in figure 5.15. These are the places in the curve at the bottom of the peak where the blue line gets thick. Looking at the peaks, the oscillations can be seen at the tail end of the peaks, right when the course difference is approaching zero. A detail of the tail end of the peak is shown in figure 5.16 below.

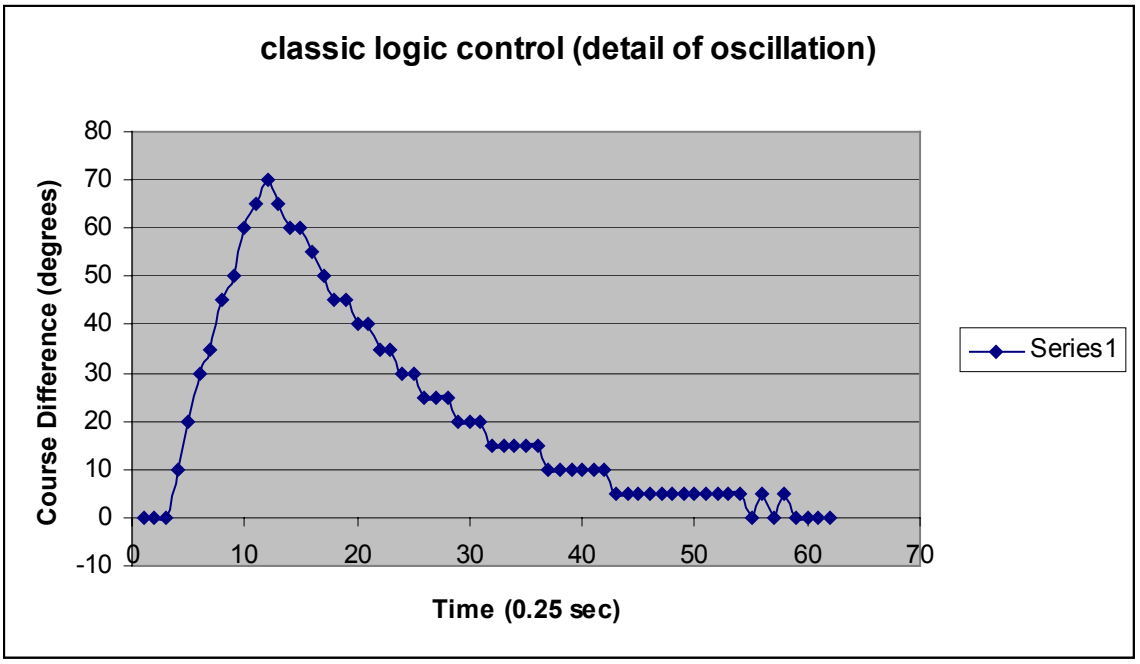

Figure 5.16. A detail of the first peak from figure 5.15. Looking at the curve at about time step 55, the oscillations can be seen as the curve moves from a course difference of 0 to 5 degrees and back twice.

Looking at figure 5.13, these oscillations are not present. Not only have these oscillations been observed in these tests, the same phenomenon occurs in the lab using the mobile robots.

Figure 5.17 and 5.18 illustrate the same type of test using the principle parts controller. Notice in figure 5.17 that the beginning of the curve shows an inter robot distance of just over 2.2 meters. Once the leader makes its turn, the follower makes up some distance, and slowly stabilizes on a following distance of about 1.8 meters. The course difference plot in figure 5.17 shows how each time the lead robot turns the follower takes time to complete the course change. This effect is seen at the tail end of each of the peaks that correspond to leader coarse changes. Also notice that the difference between the leaders course and the followers course gets up to 90 degrees, where in the earlier tests, the

largest difference seen was 70 degrees. This is due to the FFNN being non-reactive. It is putting emphasis on past values, instead of only the current sensor values.

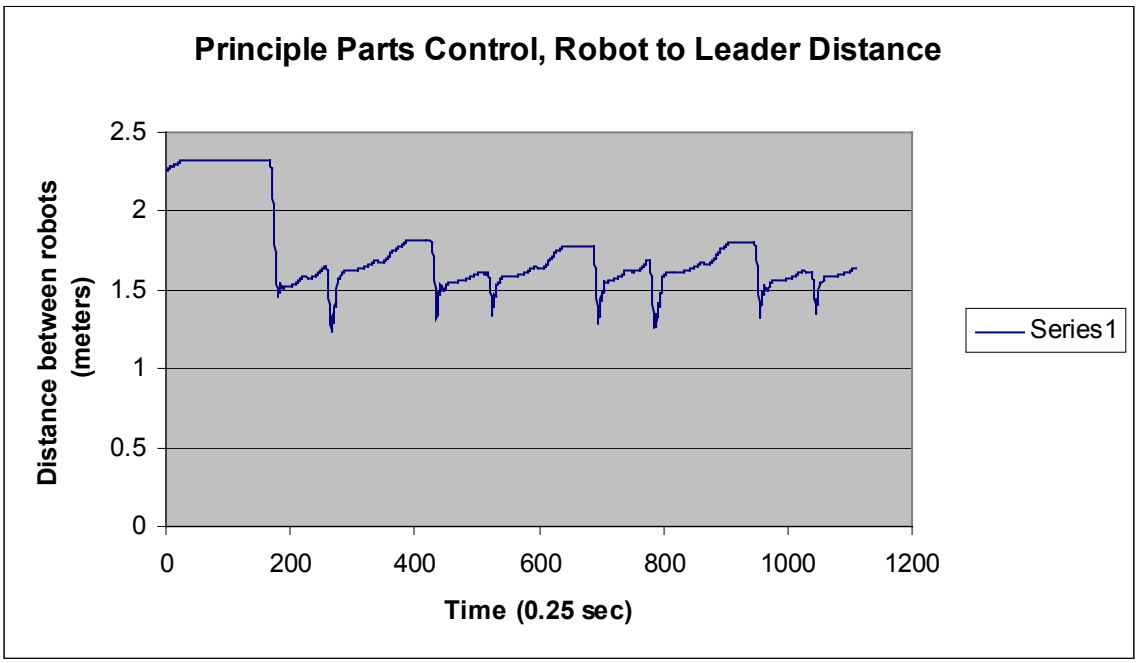

Figure 5.17. The leader/follower distance for the two lap test when the follower is using the principle parts controller.

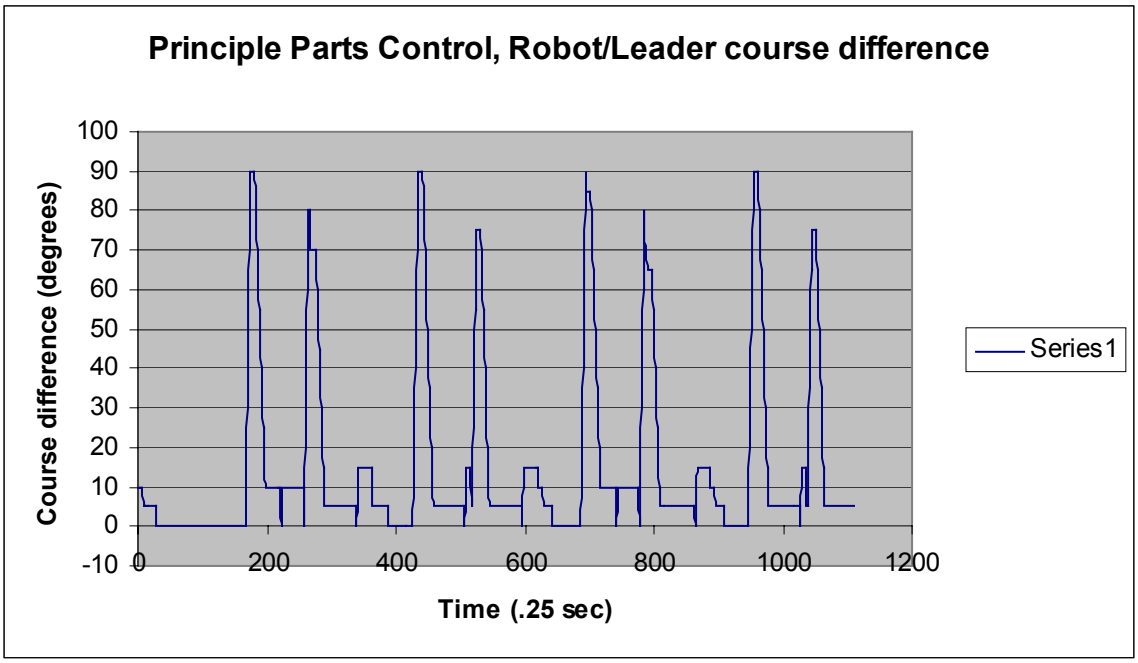

Figure 5.18. This figure shows the coarse difference between the leader and follower when the follower is using the principle parts controller. Each peak corresponds to when the leader is making a turn. Notice that this controller takes more time to complete turns than the reactive controllers.

Figures 5.19 and 5.20 show similar tests run with the behavior controller. Recall from chapter 4 that it considers past values in order to switch between seek, search, and follow behaviors. For this test it remained mostly in follow behavior, which is essentially the same as the classic logic controller, but because it did move into seek behavior many oscillations in the plots are seen. The oscillations are caused because in seek behavior the on-center zone is narrower, making the robot more sensitive to amplitude differences.

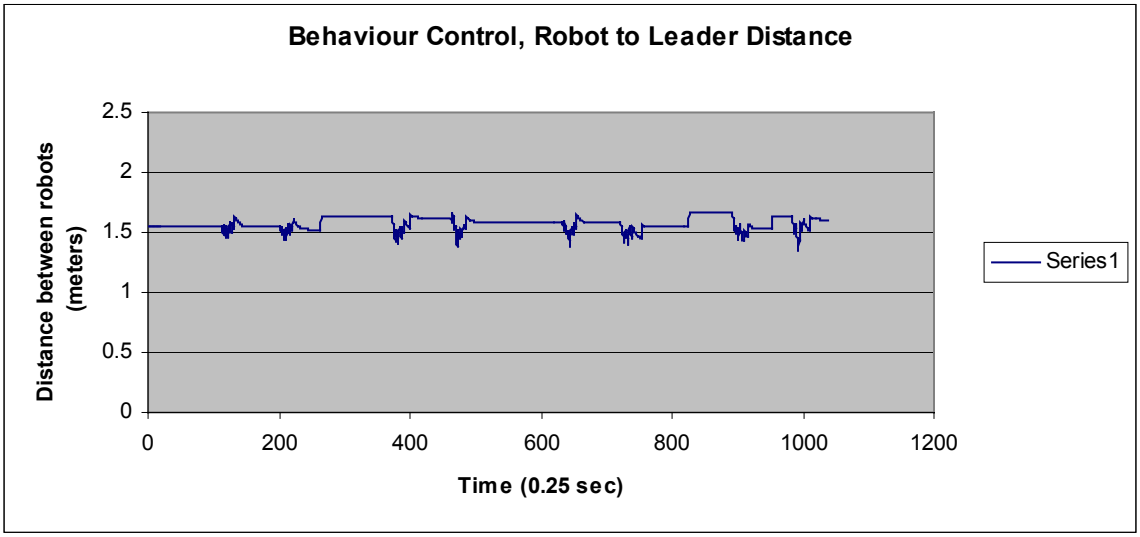

Figure 5.19. The leader/follower distance for the two lap test when the follower is using the behavior controller.

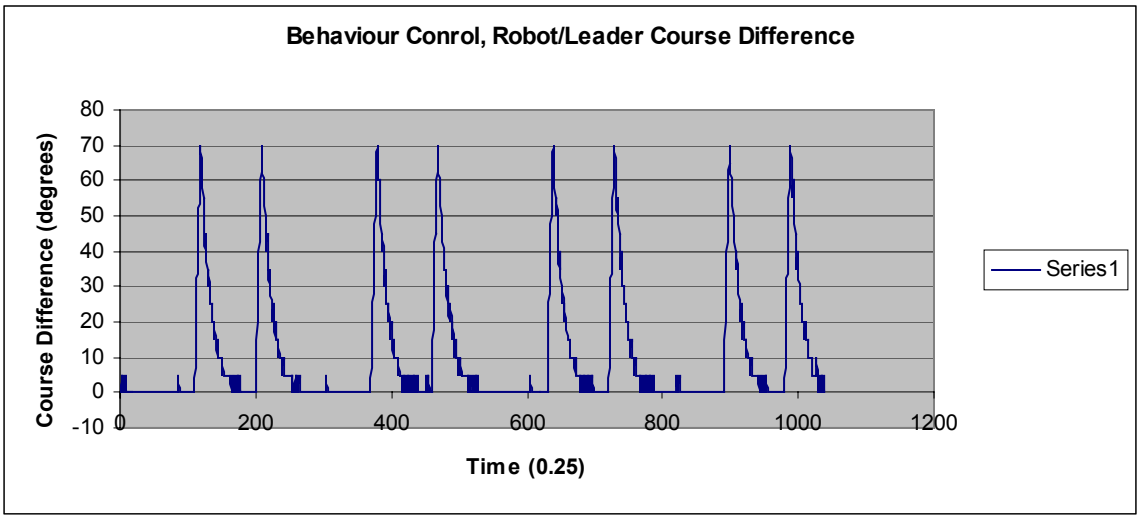

Figure 5.20. The course difference between the leader and follower when the follower is using the behavior controller. Each peak corresponds to when the leader is making a turn. Notice that this controller takes more time to complete turns than the reactive controllers. The thick blue lines at the end of each peak denote severe oscillation at the tail end of each turn. This is caused by the robot being in seek mode. In seek mode, the straight percentage parameter is set low, enabling the robot to quickly turn to its source, but at this level, it no longer damps oscillations.

Plots 5.12 through 5.20 clearly show that the performance of the FFNN controllers can match or exceed that of the hand programmed controllers. During initial testing, subjective observations indicated that the FFNN were smoother than the behavior or classic logic controllers in both the simulator and onboard the robots. This series of simulator tests backs up those early impressions. The FFNNs avoid oscillations, and can maintain stabile distances from the follower to the leader.

Tests to ascertain if the non-reactive FFNNs could solve the forward/backward ambiguity problem were also done. The tests were run in the simulator using a robot with directional capability but no forward velocity. The robot could orient itself towards the source, but not move towards it. The source was then moved directly towards the front of the robot so that if the robot did not move, the source would go perfectly between the robots left and right sensors. In the past, this test has been done many times with the baseline algorithm resulting in fooling the algorithm. If the source is driven precisely the algorithm will not react. More testing is needed but initial results show that the principle parts controller can react correctly if the source is moved quickly enough for a gradient to register in the sensor input history. If the source is moved too slowly, each of the past senor values will be the same as the current sensor value and the FFNN will not detect a drop in gradient, and will not react. While these tests showed success, it is acknowledged that further investigation is needed regarding this method of solving the ambiguity.

5.7 Summary

This chapter detailed efforts to automate the learning process. It started by discussing learning issues, and continued with an explanation of a reactive onboard learning algorithm developed for the robots using recent memories. Next, the content and acquisition of recent memories was discussed, possible solutions being sensor mirroring and the use of the random but purposeful controller. The strengths and weaknesses of the reactive approach were detailed and the case for a more capable method was made. Following that, non-reactive learning is discussed and finally results of lab and simulation testing are presented. The principle parts non-reactive method of learning has solved the forward/backward ambiguity problem, which cannot be solved by the reactive method. Unlike the method detailed by Ah-Hwee Tan et al[51], both algorithms learn from recent memories, saving the robot the trouble of having to physically execute trial solutions. The methods described in this chapter rely on assumptions and rules of thumb used at each time step so that the correct action can be found. This assumes a fairly direct cause and effect relationship between actions and sensor readings. The next chapter describes two new methods, both based on data clustering, that to some degree relaxes this assumption.

Chapter 6. Learning Using Sensor/Action Clustering and Connectivity

All the algorithms destined to run onboard the robots up to this point have been supervised algorithms. That is, for each sensor input in the training set, either a human or a program indicated which action was appropriate or inappropriate. The previous chapter detailed the various automated methods of accomplishing this. They relied on assumptions and rules of thumb used at each time step in order to find the correct action. While this is better than using a human in the loop, it is still basically a supervised learning system that relies on an assumption of a fairly direct cause and effect relationship between actions and sensor readings.

That approach is markedly different than the unsupervised technique used in the multi-robot formation maneuvering simulator. Recall that the fitness test for that simulator was based on a follower robot's average distance from a computer controlled leader robot. Here the fitness function measured the results at the end of a robot's run. The GA promoted robots that remained closest to the leader to the next generation and let them serve as parents to complete the remaining population. Instead of trying to teach the robots what actions were best based on a goal parameter, the robots in conjunction with the GA found the senor/action pairs that combined to directly satisfy the overall goal. This technique works well when the goal is known and it is hard to discern the correct action at each instance in time. The downside is that it did require that several solutions needed to be tested in simulation, which is not practical for a physical robot. What is needed is a method that can make use of the unsupervised strategy but avoid repeated physical execution with the robot.

This chapter describes two new unsupervised methods that relax the need to a-priori define specific cause and effect assumptions. Both methods use data clustering[53], a means of grouping like data items, to break down the recent memory into a collection of situations. The first method uses an unsupervised learning algorithm to train FFNNs to select correct actions based on predicted sensor readings. The second method is a computer graph[54] based method whose original purpose was to provide an observer with a situational memory. Learning using this system is accomplished by using the shortest paths to stabile states. The next section describes how clustering is used to identify the different experiences contained in a recent memory.

6.1 Recent Memory: A Collection of Situations

The learning methods in this chapter view a recent memory comprised of sensor/action pairs as a sequence of situations. Using this line of thought, a situation is a sequence of sensor/action pairs that consistently describes an event. For example when driving a car, there are situations in which the driver accelerates, stops, makes left turns, right turns, etc. A trip to the store could be defined as a sequence of these situations. Both learning methods use clustering techniques to identify these distinct situations in the recent

memory. The bullets and logic below provide an overview of these ideas, with examples and more detail in the following paragraphs and sections.

- A recent memory M, is collected using the 'purposeful but random' controller as described in 5.2. M is a sequence of sensor/action pairs, SA.
    - M = SA[0]…..SA[n-1]; n is the length of M

- A sensor action pair, SA, is defined as an ordered 4 tuple, {L, I, R, A} whose members are defined below.

    - L: left relative microphone value
    - I : sound intensity value (average of L and R)
    - R : right relative microphone value
    - A : action; The action can be left, straight, or right

- A situation, S, consists of a sequential list of sensor/action pairs from memory M such that:

    - S [i] = {SA[k], SA[k+1],…,SA[k+situation length] }
    - situation length : For these algorithms the situation length has been set at 4. Sizes varying from 4 to 16 have been used. In general the situation length should be long enough so that a trend can be seen, but not so long that several trends are present in the situation.

The algorithms in this chapter use different clustering techniques to create a library of situations. Each cluster represents a situation in the recent memory. The clustering technique used in the sensor prediction algorithm is based on the Kohonen[55] self organizing feature map (SOFM) and illustrated at the indicated web location[56]. This clustering algorithm has the unique attribute that clusters are topologically related to each other; this characteristic is used advantageously in this work in that adjacent codebook elements (the data structure that holds the cluster is referred to as an element, the cluster is analogous to a situation that occurs in the recent memory) are similar to each other and form trends leading towards or away from the goal state. The clusters that this algorithm creates are kept in a data structure commonly referred to as a codebook[57], so that term is used here. The bullets below describe the codebook data structure and its relationship to the recent memory.

- Codebook : the library of situations found in the recent memory by a clustering algorithm.

    - Codebook = C[0], C[1],…,C[Number of situations-1]. The C[j]'s are commonly referred to as codebook elements.

- M = {C[h], C[k], C[j],…,C[m]}; 0 <= h,k,j, m < Number of situations-1

To summarize, the recent memory is a sequence of sensor/action pairs collected by the robot using a controller such as the 'purposeful but random' controller. A situation is a

group of sensor action pairs that represents a sensor/action trend. The situations are found using clustering techniques. The recent memory can be viewed as a sequence of situations. The following paragraphs provide an example designed to illustrate these concepts.

Consider an example in which a person's hand is to the left of a burning candle and they move their hand a little to the right. As they do so, they will feel the warmth of the candle increase. Moving their hand a little more towards it causes more warmth, and moving away causes a little less warmth. If the person were blindfolded and their friend set the candle in front of them and guided their hand towards and away from the candle, it would not take long for the blindfolded person to figure out which direction causes their hand to warm up.

If their friend put their hand a few inches to the left of the candle and asked them what they think they would feel if they moved their hand to the right (towards the candle), the blindfolded person would immediately know that their hand would feel a little warmer than it currently is. They would not have to actually move to the right to know that, because they would recall that situation experienced earlier when their friend was guiding their hand back and forth around the candle. They would be able to compare their current feelings and the requested action to a time when their hand felt the same way and their friend moved it a little to the right and the warmth increased. They would make a prediction based on an earlier experience that was similar. The topic of prediction based on previously experienced situations is key to the operation of the sensor prediction based learning algorithm. It will be discussed later using an example in the text.

Table 6.1 shows a recent memory based on the candle/hand example in which the persons hand is initially to the left of the candle for a moment, moved to the left away from the candle, then moved right towards the candle and left there.

Table 6.2 below shows the codebook of situations, with a situation length of 4, developed from the memory described in the above example and shown in Table 6.1. Note that because of the non-deterministic nature of the Kohonen SOFM algorithm, the codebook will be populated with situations that are similar to those that exist in memory are created, are not necessarily exact representations of them.

Comparing the codebook to the recent memory, we see that the recent memory is formed of situations in the order of codes 0, 4, 5, 1 and 3.

Earlier in the candle/hand example there was a discussion about prediction based on past situations. In the example the blindfolded person was asked what they thought they would feel next if their hand was to the left of the candle and was moved to the right, towards the candle. Table 6.3 below describes this situation. The blue cells denote the situation described to the person. The yellow cell denotes the action that the person was asked about if they were in the described situation. Said another way, the question is asked if the situation is S and action A occurs what will be the outcome in terms of sensor values?

Table 6.1. A recent memory of the candle/hand example in the text

Sensor/ Action Pair Number	Hand Warmth (0 – 10) 0 is cold, 10 is hot	Action (-1 left, 0 no move 1 right)
0	5	0
1	5	0
2	5	0
3	5	0
4	5	-1
5	4	-1
6	3	-1
7	2	-1
8	2	1
9	3	1
10	4	1
11	5	1
12	5	1
13	6	1
14	7	1
15	8	1
16	8	0
17	8	0
18	8	0
19	8	0

Table 6.2. The codebook of situations for candle/hand example described in the text. The hand warmth/hand action columns comprise the sensor/action pair. As can be seen in from the table entries a move to the right warms up the hand, a move to the left cools the hand, and not moving causes no changes.

Situation, S, Number	Hand Warmth (0 – 10) 0 is cold, 10 is hot	Action (-1 left, 0 no move 1 right)
0	5	0
	5	0
	5	0
	5	0
1	5	1
	6	1
	7	1
	8	1
2	8	-1
	7	-1
	6	-1
	5	-1
3	8	0
	8	0
	8	0
	8	0
4	5	-1
	4	-1
	3	-1
	2	-1
5	2	1
	3	1
	4	1
	5	1

Table 6.3. The situation that the blindfolded person was asked about in the candle/hand example.

Sensor/ Action Pair number	Hand Warmth (0 – 10) 0 is cold, 10 is hot)	Action (-1 left, 0 no move 1 right)
t+0	5	1
t+1	6	1
t+2	7	1
t+3	?	

To answer the question, the hand warmth and action columns are compared to the first 3 columns of each situation in the codebook in table 6.2. The closest match using a Euclidean difference is the first 3 rows of situation 1 in the codebook. The implication of using a closest match instead of an exact match is that the memory does not need to contain all possible situations allowing the software some degree of gerneralization. They are shaded purple for clarity. The next row of situation 1 contains a hand warmth value of 8. If the environment is the same when the question was asked as it was when the situation library was created and the hand makes the same types of motions, then the person should expect to feel a hand warmth of about 8. This expected hand warmth is shaded green for clarity. This example is important because this is the principle that the sensor prediction based learning is based upon. The logic below describes this principle.

- A situation, S, consists of a sequential list of sensor/action pairs, SA, from memory M such that:
  - S [I] ≈ {SA[k], SA[k+1],…,SA[k+situation length-1] }
  - SA is formed of sensor/action pairs
    - SA→SV denotes the sensor values in the sensor/action pair.
    - SA→A denotes the action value in the sensor/action pair.
- Codebook : the library of situations found in the recent memory by a clustering algorithm.
  - Codebook = C[0], C[1],…,C[Number of situations-1]. The S[j]'s are commonly referred to as codebook elements.
- Z0 is a training situation formed as follows:
  - P : proposed situation that is being asked about (as in the example above).
    - P = {SA[0], SA[1],…,(SA[situation length – 2]→SV)}
  - A: given the proposed situation P, A is the action combined P whose result is in question. Said another way, the question is asked, if P is the situation and A is the selected action, what is the predicted result?
  - C[i]→SA[situation length-1] is the last sensor action pair from the codebook element that most closely matches P+A using a Euclidean difference.
    - C[i]→SA[situation length-1] →SV are the sensor values that complete the situation. Since Z0 did not necessarily come from a situation directly experienced in the recent memory C[i]→SA[situation length-1] →SV are referred to as predicted sensor values.
  - Collecting the variables:
    - Z0 = {SA[0], SA[1],…,(SA[situation length – 2]→SV, A), C[i]→SA[situation length-1]}

The above logic relies on following assumption:

Z0 ≈ C[k] ; k is the index of the situation in the codebook that most closely matches Z0.

The above assumption along with the constraint that the environment does not change so quickly that the elements (situations) in the codebook become outdated before learning occurs are key to both the algorithms in this chapter. How often the codebook and network get rebuilt are dependent on how quickly the environment changes. The next section describes the sensor prediction based learning process that relies on the principles set forth in this section.

6.2 Unsupervised Learning Using Sensor Prediction

The unsupervised learning algorithm using sensor prediction is based on using a GA to train a neural network to select the actions that gain sensor intensity for the entire set of situations, S[0],…,S[number of situations –1], encountered in the recent memory. The process is outlined below.

- The recent memory M is collected using a controller such as the 'purposeful but random' controller.
- The Kohonen SOFM algorithm is used to create a codebook of situations from M.
- The GA is used to train a FFNN by promoting networks that consistently select actions that gain sensor intensity for the entire set of situations in the codebook. This process is described in detail in this section.

Chapter 5 discussed the operation of the purposeful but random controller. The memory M that it collects is arranged as described in the previous section. The Kohonen SOFM uses this memory to create a codebook of situations. The recent memory, the codebook size (number of situations), and codebook element length (the same as the situation length) are the inputs to the Kohonen SOFM algorithm. The Kohonen algorithm is an unsupervised competitive learning algorithm with a random element to it that can be used to cluster[53] data. The output of the algorithm is a codebook of the requested size. The individual codebook elements are used to represent the situations in the memory. The codebook that this algorithm creates is effectively a set of approximate situations. The difference between the codebook and situations made from the original memory by arbitrarily selecting sequences of SA pairs is that the codebook provides a superior representation of the memory because it covers the situations in memory and those similar but not necessarily encountered.

The SOFM algorithm has the characteristic that the elements near each other (those whose indices are close numerically) in the codebook are similar to one another. In this work similarity was measured with both correlation coefficients and Euclidean differences. The majority of the work was done using the Euclidean difference because it gave approximately the same results and it is computationally less expensive.

The number of codebook elements requested regulates the amount of similarity between the codebook elements. A codebook with few elements will have elements that are not similar to one another, and conversely, a codebook with many elements will have elements that are very similar to one another. For this application the codebook size was

selected to be relatively large, usually 128 elements, with an element length ranging from 4 to 16. These parameters were found empirically by using the codebook elements to compress the recent memory, and then compare the reconstructed version against the original. With more codebook elements, the reconstructed memory more closely matched the original. This process is similar to that of a lossy vector image compression. Though this process was done by hand, it could be automated.

Selecting the situation length and number of codebook elements parameters that result in an accurate memory reconstruction helps ensure that the assumption below is true:

$Z0 \approx C[k]$; k is the index of the situation in the codebook that most closely matches Z0.

With few elements in the codebook, and a large codebook element size this assumption, that Z0 very closely approximates the closest match C[k], will not be true.

In the previous section the candle/hand example was used to describe the expected sensor values given a proposed situation and action. The GA uses a process very similar to this to retrieve sensor intensity values that it uses in its fitness function when training formation maneuvering controllers. The process is detailed by the logic below.

The memory and codebook are defined as:

- The recent memory M, which was collected using the 'purposeful but random' controller as described in 5.2. M is a sequence of sensor/action pairs, SA.
    - M = SA[0]…..SA[n-1]; n is the length of M

- A sensor action pair, SA, is defined as an ordered 4 tuple, {L, I, R, A} whose members are defined below.

    - L: left relative microphone value
    - I : sound intensity value
    - R : right relative microphone value
    - A : action; The action can be left, straight, or right

- A situation, S, consists of a sequential list of sensor/action pairs, SA, from memory M such that:
    - $S[i] \approx \{SA[k], SA[k+1],\ldots,SA[k+\text{situation length}-1]\}$
        - situation length - For this algorithm it has been set at 4. In general the situation length should be long enough so that a trend can be seen, but not so long that several trends are present in the situation. This issue is discussed in the above text.
    - SA is formed of sensor/action pairs
        - SA→SV denotes the sensor values in the sensor/action pair.
        - SA→A denotes the action value in the sensor/action pair.

- Codebook : the library of situations developed from the recent memory by the Kohonen SOFM algorithm.
    - Codebook = C[0], C[1],…,C[Number of allowed situations-1]. The C[j]'s are commonly referred to as codebook elements.
        - Number of allowed situations : The number of situations is an adjustable parameter. Its usual value was 128, which was found empirically, but could be automated using a compression method similar to the method suggested earlier. This issue is discussed earlier in this section.

Once the codebook is available to the GA, the GA's fitness function measures the effectiveness of the controllers in the population by testing each of them with each situation (codebook element, C) in the codebook. The test is similar to the process outlined in the previous section for finding the predicted sensor values. The logic below details the process used by the GA's fitness function.

Energy : A variable that sums intensity across the codebook.

Energy = 0;
For (each C[j] in the codebook, j = 0 to (number of allowed situations – 1)) {

- Z0 is a prospective situation formed as follows:
    - P : proposed situation that a FFNN controller is being asked about.
        - P = {C[j] →SA[0], C[j] →SA[1],…,(C[j] →SA[situation length – 2]→SV)}
    - A: given the proposed situation P, A is the action generated by the FFNN controller.
    - C[i]→SA[situation length-1] is the last sensor action pair from the codebook element that most closely matches P∪A using a Euclidean difference.
        - C[i]→SA[situation length-1] →SV are the sensor values that complete the situation. Since Z0 did not come from a situation directly experienced in the recent memory C[i]→SA[situation length-1] →SV are referred to as predicted sensor values.
    - Collecting the variables:
        - Z0 = {P, A, C[i]→SA[situation length-1]}

- Energy = (Z0→SA[situation length-1] →I) –
  (C[i]→ SA[situation length-2] →I);

} /* End for. */

The last statement determines if the intensity resulting from the action generated by the candidate FFNN is greater than the intensity generated by the closest code book match to the predicted situation. If it is greater then the FFNN did better and is rewarded. Using the above process the entire population of controllers is ranked and the GA promotes the

controllers that gain the most energy, i.e. those with the highest value for the Energy variable, for the specified number of generations.

Cumulative energy gain is used as the fitness metric in the following robots because it is proportional to the inter-robot distance. As the inter-robot distance decreases, the intensity increases at the robots sensors.

The algorithm works much like the unsupervised technique used in the multi-robot formation maneuvering simulator. Recall that the fitness test for that simulator was based on a follower robot's average distance from a computer controlled leader robot. Here the fitness function measured the results at the end of a robot's run. Instead of using average distance at the end of a set number of steps as a measure of fitness, this algorithm uses energy gain at the sensors. The energy gain, or loss, is a sensor based measurement of distance, larger values indicating closer to the signal source, and smaller values meaning further away.

This algorithm was used to train FFNNs that controlled robots in formation maneuvering tasks from data collected in the simulator and from data collected in the lab using mobile robots. Sensor mirroring was used at both the raw sensor value level and at the codebook level. Mirroring at the codebook level is implemented in a similar manner to mirroring at the raw level, with the difference that the entire codebook element is mirrored. This helps ensure consistency since the Kohonen SOFM always returns a different codebook because of its random nature. The advantage of this approach is that it is unsupervised, thus no step-by-step assignment of correct actions is made, and the algorithm does not require the robot to make multitudes of trial runs in order to learn. Results are shown later in section 6.4.

The next section describes an algorithm that also uses clustering to divide the recent memory into a library of experiences. This routine uses a graph structure to provide a basis learning.

6.3 Unsupervised Learning Using a Directed Graph

So far, all the algorithms discussed have used either a supervised training set or recent memories to produce a FFNN. Once that FFNN is produced the recent memory is not used. The method presented in this section uses a directed graph, created from a recent memory, to create a training set that is used by a GA to create a FFNN that can be used to control a robot. This section starts by describing the general idea behind using a graph as a situational memory and the benefits that it provides. Following that is a description of how the graph is created, and then finally the section closes by giving a detailed description of how the graph structure was used to train a FFNN for use in a simulated following robot.

In this method of learning a recent memory is viewed as a sequence of situations that occurred over time. In that recent memory there are situations that are more desirable than others. The situations are expressed as groups of sensor/action pairs as is the goal. Desirable situations are those whose sensor/action pairs are similar to the goal's

description of the ideal sensor/action pairs. Learning is based on finding the shortest sequence of situations from each less desirable situation to the most desirable situation.

The general idea is to create a directed graph[54] that describes how the situations in the recent memory are related to one another. The connectivity of the directed graph can be used to determine which situations usually follow each other and what actions lead to what situations. Once this information is known, a situation that represents the goal can be selected, whether it is by the user, or by automatic means. The graph is the tool that is used to find the progression of situations from less desirable situations to the goal situation. Using this information, an effective controller can be constructed by associating the sensor readings in less desirable situations with the actions that were done at the time that lead to more desirable situations.

The graph's connectivity stores the topological relationship between the different situations encountered in the recent memory, and is the key difference between this method and Q Learning. As described earlier in chapter 2, Q Learning creates a table of states and recommended actions for each state, but information of how a sequence of states and actions relate to one another is not used or preserved. This relationship between the situations is the key to the functionality of the graph algorithm. Using these relationships a learning algorithm based on the shortest path to the desired state was developed. The next paragraphs provide an overview of the algorithm.

The graph learning algorithm relies on the following key processes in order to create a training set for the GA to generate a non-reactive FFNN. The bullets below outline these processes, and the sections thereafter provide more detail on the methods used to implement these processes.

- The recent memory M, which was collected using the 'purposeful but random' controller as described in 5.2, is separated into a library of situations. For example when driving a car, there are situations in which the driver accelerates, stops, makes left turns, right turns, etc. A trip to the store could be defined as a sequence of these situations. A more precise description of how the situation/memory relationship is given below for a formation maneuvering task.

  SA : sensor/action pair consisting of the ordered 4 tuple {L, I, R, A}whose members are defined below.

  L: left relative microphone value
  I : sound intensity value
  R : right relative microphone value
  A : action; The action can be left, straight, or right

  M : recent memory consisting of sensor/action pairs SA[0] …. SA[n-1]

  S : situation, consisting of a sequential list of sensor/action pairs from memory M such that:

S [i] = SA[k], SA[k+1],…, SA[k+situation length];
k >= 0 and k < (n – 1 –situation length)

situation length : For this algorithm it has been set at 4. In general the situation length should be long enough so that a trend can be seen, but not so long that several trends are present in the situation. This issue is addressed in section 6.2. As before the values were arrived at empirically but can be found automatically using the memory compression method suggested earlier.

Codebook: the library of situations found in the recent memory. Note that for this algorithm the codes are actual situations from the memories and not approximate situations as with the Kohonen algorithm.
Codebook = C[0] …..C[Number of situations-1], where C[i] is an individual code.

Number of situations: The number of situations is an adjustable parameter. As in the previous section its approximate value was found empirically, but could be automated using a compression method similar to the method suggested earlier. We say approximate because the clustering routine used in this algorithm dictates the number of clusters. As before the clusters represent the situations in the recent memory.

M = {C[h], C[k], C[j], …….,C[m]}; 0 <= h,k,j, m < Number of situations

- A situation C that represents the goal state, **G**, is identified. In the robot following task this state is one in which the robots sensors are reading about equal, the intensity is near the average of the high and low intensities experienced and the robot is not changing its course. Using the nomenclature from section 5.5.1:

  For each SA[0]…SA[situation length] $\in$ **G**,
  (SA$\rightarrow$L $\cong$ SA$\rightarrow$R) and
  (SA$\rightarrow$I $\cong$ average intensity value) and
  (SA$\rightarrow$A = straight)

  Note that the above implies that any given memory may contain multiple goal states.

- The order in time that the situations occurred in the recent memory is found. This step provides information on the frequency that the various individual situations occurred. It also provides information on which situations usually follow each other. A more precise description of the memory/situation relationship is given below:

  $\forall$ C[i] $\in$ Codebook

M = {C[1], C[2], C[4], C[1], C[2] …….C[4], C[1]}

- Using the connectivity established in the above step, the situations that lie on the quickest route from non goal states to the goal state **G** are used to create a training set for the a GA process that generates a non-reactive FFNN. In the description below the symbol ⇒ indicates that there is a path from C[i] to C[j].

  For each C[i] in the Codebook find the quickest route to **G**:

  C[0]⇒C[i]….C[j] ⇒ **G**
  C[1]⇒C[m]….C[n] ⇒ **G**
  .
  .
  .
  C[number of situations – 1] ⇒….C[q] ⇒ **G**

  Let F be the set of all situations, C[k], that lie on one of the quickest routes to **G**.

  i.e., C[k] ∈ F if C[k] lies on a shortest path to **G**.

  Creation of the set F is thus used to sequentially filter data from the recent memory M into a training set that is written out in the same format as that of the principle parts routine described in the previous chapter. From this point the GA routine is used to create a non-reactive FFNN.

It is worth noting that in this learning algorithm the values of the sensor/action pairs contained within a situation are not used to determine if that situation will be used for training the FFNN. The metric that is used to determine if a situation is to be included in the training set is whether or not that situation is similar to a situation that lies on one of the shortest paths to the goal state. This method is not based on a process that compares the values of SA[i]→I and SA[i+1]→I to each other. For this reason, this algorithm is called unsupervised.

The next sections provide details on the bullets from the above discussion and how the situation connectivity established by these processes is used learn. It details how the graph is created from the recent memory. First, an overview of the algorithm is given, followed by a description of the clustering algorithm, then the compression technique that finds the connectivity between the situations encountered in the recent memory is detailed. Following that the use of Floyd's[58] algorithm to find the paths to the desirable states is discussed and compared to the Q Learning algorithm. And finally a brief review of results obtained using the graph method for learning is given.

6.3.1 Creating the Cluster Graph

As with the sensor prediction based method discussed in the previous section, the graph based technique creates a library of situations from recent memories by using clustering. Instead of using a SOFM, a Radial Basis Function[59] (RBF) clustering method is used. The radial basis function is used because the amount of similarity (closeness of the

clusters) can be precisely controlled with the radial basis function distance. In this way the degree of similarity between the clusters can be controlled using a quantitative measure, the RBF distance. It also has the advantage that it will produce the same codebook each time because it does not contain a random element as does the Kohonen SOFM procedure. In the RBF clustering algorithm below, the closeness of the clusters is regulated by the comparison done by the statement that compares the RBF distance to the constant 0.9. Note that for the RBF clustering algorithm the resulting codes are actual situations from memory, whereas the codes resulting from Kohonen clustering were approximate situations.

The RBF clustering algorithm variables are defined as follows:

- cluster_size: the cluster size is set to the situation length as described earlier.
- cluster_set_count: Total number of RBF clusters in the recent memory M, where this number is dependent upon the RBF distance specified.
- cluster_set: The set of RBF clusters contained in M.
- P: Pointer to the current place in the recent memory M.
- DV: data vector comprised of situation length consecutive SA pairs from M
- eDist: Euclidean distance between DV and cluster in cluster_set
- Codebook: the library of situations found in the recent memory.

The algorithm is shown below:

```
cluster_size = situation length;
cluster_set_count = 0;
cluster_set = ∅;
P = 0;

While (P has not passed the end of the recent memory) {
        DV = cluster_size sensor/action pairs from point P in the recent memory.

        /* If the cluster set is empty. */
        If (cluster_set_count is 0) {
                /* Add the data as the first cluster. */
                DV ∪ cluster_set;

                /* Increment cluster count. */
                cluster_set_count = cluster_set_count + 1;
        }
```

```
        /* If clusters have already been added. (cluster_set_count > 0) */
        else {
                /* Find the distance from DV to the closest cluster in cluster_set */
                eDist = min(Euclidean distance(DV,
                                                cluster_set[0…cluster_set_count -1]);

                /* Find the RBF distance. */
                RBFdist = exp(-1 * eDist^2);

                /* If the cluster is not close to any existing clusters. */
                If  (RBFdist < 0.9 ) {
                      /* Add the data as a new cluster */
                      DV ∪ cluster_set;

                      /* Increment cluster count. */
                      cluster_set_count = cluster_set_count + 1;
                }
        }
         /* Increment index P by cluster_size */
        P = P + cluster_size;
} /* End while */
Codebook = cluster_set;
```

The result of the algorithm is a set of clusters that is cluster_count in length. When the Euclidean distance between the current cluster and its closest neighbor is close to zero, the RBF distance parameter is near one, see figure 6.4. As the Euclidean distance grows, the RBF distance shrinks. By creating new clusters when the RBF distance is high (0.9 for example) the clusters are spaced closely to one another, conversely, if new clusters are created only when the RBF distance is low (0.5 for example) the clusters will have more separation between them. The graph below shows the relationship between the Euclidean cluster distance and RBF value.

Once the memory has been broken down into a group of clusters, the connectivity of the situations that the clusters represent is found by using the clusters to compress the recent memory. This compression process is very much like what happens in a lossy image vector compression. In this type of compression the raw data of the image is separated into finite sized groups, usually referred to as data vectors. Each data vector is compared to a codebook of vectors that has been created to compress the image. The index number of the codebook member that is the closest to the data vector is transmitted instead of the data. In this type of compression, the codebook is transmitted, followed by the indices that consist of the image. As an aside, an ideal codebook will contain the set of vectors that is most statistically common within the image. For example, if a scene that had a green grassy field were to be compressed, the codebook would contain several vectors (sometimes called dither patterns) that could represent the various colors and textures of the grass. This codebook would be fine for grassy field compressions, but would not be

good for compressing the text on this page because the dither patterns in the codebook would be so different from the text image. The compression algorithm is shown below.

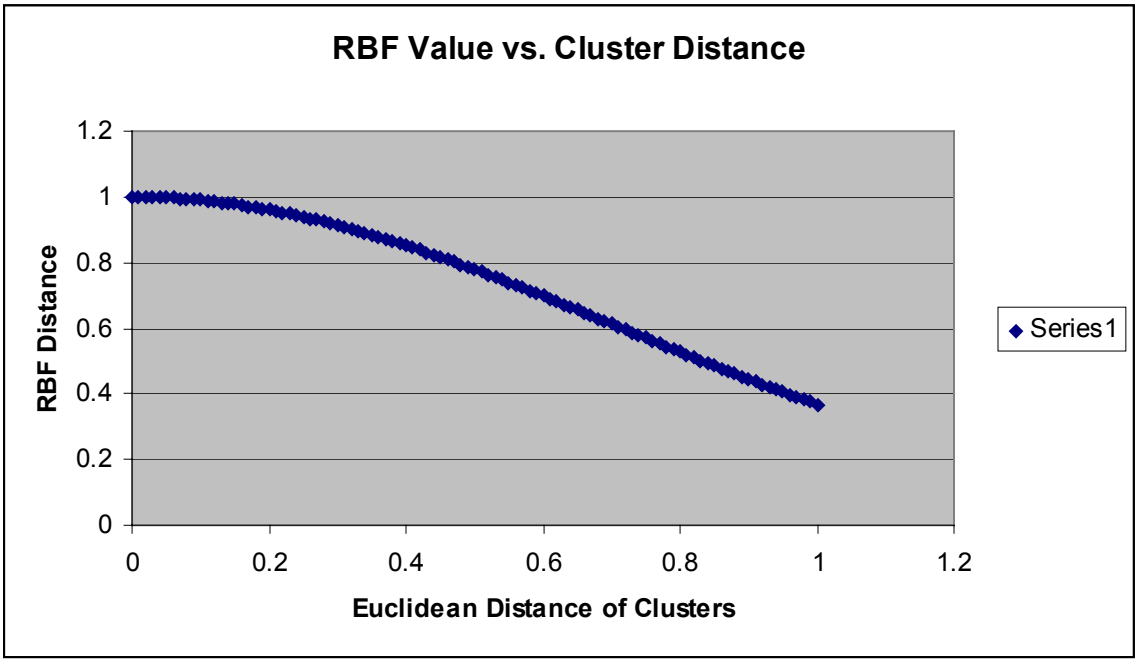

Figure 6.4. The relationship between RBF distance and Euclidean distance.

The variables are defined as follows:

- M: recent memory consisting of sensor/action pairs SA[0] …. SA[n-1]
- S: S is a cluster, consisting of a sequential list of sensor/action pairs from recent memory M such that: S [I] = SA[k]….SA[k+cluster_size]; k >= 0 and k < (n – 1 – cluster_size)
- cluster_size : The size of the clusters in the codebook. It is set to the situation length as described earlier.
- Codebook: the library of situations found using the above algorithm Codebook = C[0] …..C[number_of_clusters-1].
- P: Pointer to the current place in the recent memory M.
- DV: data vector comprised of situation length SA pairs.
- closeIndex: index of the closest cluster to DV.
- Cluster_indices: list of cluster indices.

- NumIndices: length of Cluster_indices list.

The algorithm is shown below:

```
P = 0;
Cluster_indices = ∅;
NumIndices = 0;
While (P has not passed the end of the recent memory M) {
        DV = cluster_size sensor/action pairs from point P in the recent memory;

        Set closeIndex to the index of the closest cluster in cluster_set to DV;

        Append CloseIndex to Cluster_indices;
        NumIndices = NumIndices + 1;
        P = P+cluster_size;
} /* End while. */
```

If the cluster set contained 5 clusters, each with a cluster size of 10, and the recent memory had 100 SA pairs the resulting list would have 10 cluster indices in it. Below is an example list of cluster indices:

Cluster_indices = {0, 1, 2, 4, 3, 0, 1, 4, 3, 0}

While the use of this compression is unlike that of the image compression, using this technique and relying on the one dimensional aspects of time, questions like "Which experience usually follows this experience that I am interested in?" can be answered by tracing the sequence of clusters that are used to compress the recent memory.

Table 6.5. An adjacency matrix for the Cluster_indices example above. By looking at the node numbers in the leftmost column it can be seen that cluster 1 follows cluster 0 in the example, as indicated by the 1 in column 1, row 0 of the table.

Cluster number	0	1	2	3	4
0	0	1	0	0	0
1	0	0	1	0	1
2	0	0	0	0	1
3	1	0	0	0	0
4	0	0	0	1	0

As indicated previously, in the case of compressing a recent memory, the cluster list can be likened to a sequential list of situations that was encountered in the time that the memory was collected. From the cluster_indices example above, it can be seen that the

situation represented by cluster 1 follows the situation represented by cluster 0 twice in the memory.

Using the sequence of clusters produced by the compression, an adjacency matrix can be formed. The adjacency matrix is a data structure that indicates which nodes in a graph are connected to each other. Table 6.5 above shows the adjacency matrix for the Cluster_indices in the previous example above.

The adjacency matrix is formed using the following algorithm.

The variables are defined as follows:

- adj: square matrix of size number_of_clusters × number_of_clusters
- Cluster_indices: list of clusters produced by the compression algorithm described above.
- NumIndices: size of Cluster_indices set.
- j: counter
- current_cluster: cluster index
- next_cluster: cluster index

The algorithm is shown below:

```
/* Initialize the adj matrix to zeroes. */
adj[0 – number_of_clusters –1] [0 – number_of_clusters –1] = 0;

/* Initialize variables. */
current_cluster = cluster_indices[0];
next_cluster = cluster_indices[1];
j = 1;

/* Populate adjacency matrix. */
While (j < NumIndices) {
        adj[current_cluster][next_cluster] = 1;
        current_cluster = next_cluster;
        j = j+1;
        next_cluster = cluster_indices[j];
}
```

As previously stated, the adjacency matrix is the data structure that represents the basic directed graph. From this data structure several graph algorithms, including Floyd's shortest path algorithm can be run. The next section details the learning process and how Floyd's algorithm plays a key role.

6.3.2 Learning Using Paths from the Cluster Graph

Learning using the graph relies on finding the most direct path from less desirable situations to more stable and desirable situations. The paths can then be traced and the

progression of states can be used to train a FFNN. Clusters that make up the paths from the lesser states are used to filter raw data from the recent memory data to form a training set. By belonging to a cluster that is on a path to success, data is declared worthy of being used to train a FFNN, and is added to the training set. The training set is comprised of sequences of data from the recent memory in which the action is consistent and the data falls within the group of "filter" clusters identified earlier. This training file creation process is similar to the process detailed in chapter 5 except that there are no rules of thumbs being applied along the way. A completed training set is then used in conjunction with the GA to create a FFNN in a process that is similar to what was described in earlier chapters. The process is summarized below.

1. A recent memory is collected using the 'purposeful but random controller'.
2. The recent memory is clustered resulting in a codebook. The codebook elements are the individual clusters. The clusters are analogous to situations.
3. The recent memory is then compressed using the codebook so that the progression of situations in the recent memory can be traced. The progression of situations is stored as a list of codebook indices.
4. From the list of indices produced by the compression, an adjacency matrix is formed. The adjacency matrix is a data structure that describes a directed graph. This graph describes which situations follow each other.
5. A cluster from the codebook is selected to represent the goal state **G**. In the robot following task this state is one in which the robots sensors are approximately equal, the intensity is near the mean of the range of intensities experienced while the robot is not changing its course.
6. Graph algorithms are used to find the shortest paths from all clusters whose energy is less than that of **G**.
7. The clusters used in the shortest paths are put into a set of desirable clusters.
8. Raw data from the recent memory that is in the set of desirable clusters is used to create a training set.
9. The training set is used along with the GA to create a FFNN as described in earlier chapters.

The first four items are described in the previous section. The following paragraphs provide more detail on items 5 through 9.

The selection of **G** is crucial to the success of this strategy. If **G** is not consistent with goals then the wrong functionality will be learned. Also **G** must be obtainable, and well represented in the recent memory. For example, an extreme high sensor energy/turning right state would not be a good goal state for a following robot. The best state would be one in which the robot is not turning, and its sensors read a balanced mid level of energy. This is the state the robot is in when it is locked on to its leader at a maintainable distance. In the work done for this dissertation the researcher selects **G** from the clusters in the codebook. Once the goal state is identified, the paths from other less desirable states need to be found.

The paths from less desirable states to the goal state can be found using a standard graph algorithm such as Floyd's algorithm. Floyd's algorithm is a dynamic programming[60] method that finds the shortest path from every node to every other node. The algorithm is shown below.

Floyd's Algorithm

The variables are defined as follows:

- number of nodes: the number of clusters in the codebook of situations created from the recent memory M.
- adj: adjacency matrix created from compression process.
- m: square matrix that is of size number of nodes × number of nodes.
- i, j, k: indices

The algorithm is shown below:

```
/*  Initialize each m[i,j] with the distance between node i, j. */
For(I = 0 to number of nodes –1) {
        For(j=0 to number of nodes –1) {
                if (i  ≠ j) {
                        m[iI][j] = adj[i][j];
                }
                else {
                        m[i][j] = 0;
                }
        }
}

/* Solve for shortest paths. */
For (k = 0 to number of nodes – 1) {
        For (i = 0 to number of nodes – 1) {
                For (j = 0 to number of nodes – 1) {
                        m[i, j] = min { m[i, j], m[i, k] + m[k, j] }
                }
        }
}
```

Figure 6.6 below shows the output of the algorithm for the adjacency matrix shown in table 6.5 along with a graph of the clusters. Note that the 3rd nested for loop in the shortest path section is remarkably similar to the comparison statement (statement 4a) used in the Q Learning algorithm from chapter 2, and again shown below.

Cluster number	0	1	2	3	4
0	0	1	2	3	2
1	3	0	1	2	1
2	3	4	0	2	1
3	1	2	3	0	3
4	2	3	4	1	0

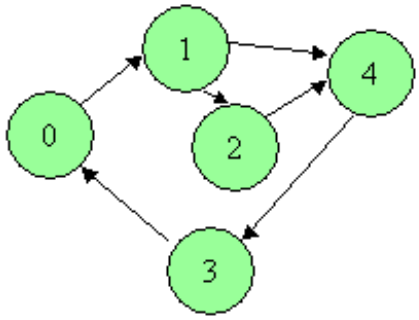

Figure 6.6. The distances of the shortest paths from every node to every other node using Floyd's algorithm. The data adjacency matrix used for this example is table 6.5. Notice the diagonal of the table is zero, indicating that there is no cost for staying in a node.

Q Learning Algorithm

s - current state
s' - the state in the set of states adjacent to state s that contains a'. (s', a') is the optimal state/action combination adjacent to s.
a-current action
a' – action that returns the greatest reward at state s'
d-amount that rewards are discounted, range is a real number between 0 to 1
r – immediate reward available for current state
Q'(s,a) – estimate of Q value at state s with action a.

For each (s, a) initialize the table entry Q'(s, a) to zero
Observe the current state s
Do forever:

1. Select an action a and execute it
2. Receive immediate reward r
3. Observe new state s'
4. Update the table entry for Q'(s, a) as follows:

    a. Q'(s, a) = r + d***max** Q'(s', a')

    This statement is adding the immediate reward available at state s to maximum reward that is available for all actions at state s'. State s' is the state that results from doing action a at state s. So this statement is taking the place of the inner loop of Floyd's algorithm. Note in this algorithm reward is maximized as opposed to path length being minimized in Floyd's algorithm.

5. s = s'

Effectively both algorithms are using dynamic programming methods to iteratively optimize their target functions. The main difference is that the result of the Q Learning algorithm is a table of states and scores for each action at each state. It does not have state-to-state transition information as does Floyd's algorithm. Floyd's algorithm not only finds the length of the shortest path from every node to every other node, it also can be used in conjunction with a path matrix so that the individual nodes along each route can be extracted.

By clustering the recent memory into groups of sensor/action pairs, the paths from lesser states to the desired state represent the best way to get from the lesser states to the desired state that has been experienced so far. It is possible that more direct routes exist between the states, but if they have not been experienced yet in the recent memory, the adjacency matrix will not contain the data to represent them. Another detail of interest is that the clustering for learning uses sensor/action pairs. An alternate approach that was tested used only the sensor readings at the node level and labeled the node-to-node connections with actions. While this approach is useful for predicting what new sensor reading will occur given a particular state and action, it introduces more complexity than needed in regard to the learning process.

The paths themselves are the key focus in learning using the graph. In the case of the following robot, a typical shortest path will start off at a low intensity level and be turning one direction or the other. Each of the subsequent nodes typically will have a higher intensity level and will be turning the same direction as the starting node. Then within one or two nodes before the goal node, a node similar to the goal node, but with less energy appears, often followed by another with still higher energy, and then finally the goal node is reached. Figure 6.7 below shows a graph for a recent memory. This graph was formed using the clustering, compression, and adjacency matrices discussed in the previous section.

Looking at the graph in figure 6.7 it can be noted that shortest path from state 3 to the goal state, 0, is as follows:

Path [3 to 0] = {3, 2, 1, 0}

Similar paths exist from states 6 and 8 to state 0. They are shown below.

Path [6 to 0] = {6, 10, 1, 0}
Path [8 to 0] = {8, 9, 1, 0}

Once the paths have been formed, the nodes that comprise them are used as a "filter" to create training examples from the recent memory. The filter derived from the graph in figure 6.7 is shown below:

Filter Clusters = Path [3 to 0] $\cup$ Path [6 to 0] $\cup$ Path [8 to 0]
Filter Clusters = {3, 2, 1, 0, 6, 10, 8, 9}

These clusters are deemed to be good clusters because, as stated earlier, each one is part of the most efficient route to the stabile state. Data that fits into this set of clusters is defined as usable data. Clusters 4, 5, and 7 were not put into the filter because they were not on a shortest path to the goal state, 0. Being included in the filter is purely a matter of being on a shortest path to the goal state, which is determined using Floyd's algorithm.

In this example it is easy to see that cluster 4 would not be a good training cluster because it would try to teach the FFNN to turn left when the sound source is closer to the right sensor (in this case L = 7 which is less than R = 10, so the source is closer to the right senor). The wrong decision in cluster 4 leads to a lower intensity value in cluster 5. In fact, this path was taken in the purposeful but random controller, demonstrating that this algorithm can be more effective. Clusters 5 is similar to 4, and 7 illustrates the same concept except the direction is different.

Identifying data to be put into the training set is done by looping through the recent memory and extracting sequences of data that are consistent in action and fall within the set of "usable" clusters. Using this scheme a training file would contain a finite number of examples. Within an example, the action is consistent. Only data that falls within the clusters in the "filter" is used. As soon as the action changes or data is found that the filter rejects, the example is ended. This process is done until all the data in the recent memory is evaluated. An outline of the algorithm is shown below.

The variables are defined as follows:

- M : recent memory
- P : Pointer into M
- SA : Sensor/action pair.
- DV : data vector comprised of a cluster size list of sensor/action pairs from M.
    - DV = {SA[P], SA[P+1], ….SA[P + (cluster size-1)]}
- codebook : the library of situations found using the RBF clustering algorithm .
- codebook = C[0] …..C[number_of_clusters-1].
- closeIndex : index that points at a cluster in the codebook
- Filter Clusters : the set of clusters that lie on the shortest paths to the goal state **G**.

The algorithm is shown below

```
P = 0;
While (P has not passed the end of the recent memory M) {
        DV = cluster_size sensor/action pairs from point P in the recent memory;
        closeIndex = index of the closest cluster in codebook to DV;
        If ((all the actions in DV are the same) and (closeIndex ∈ Filter Clusters)) {
                Write DV to file as a usable example;
        }
        P = P+1;
} /* End while. */
```

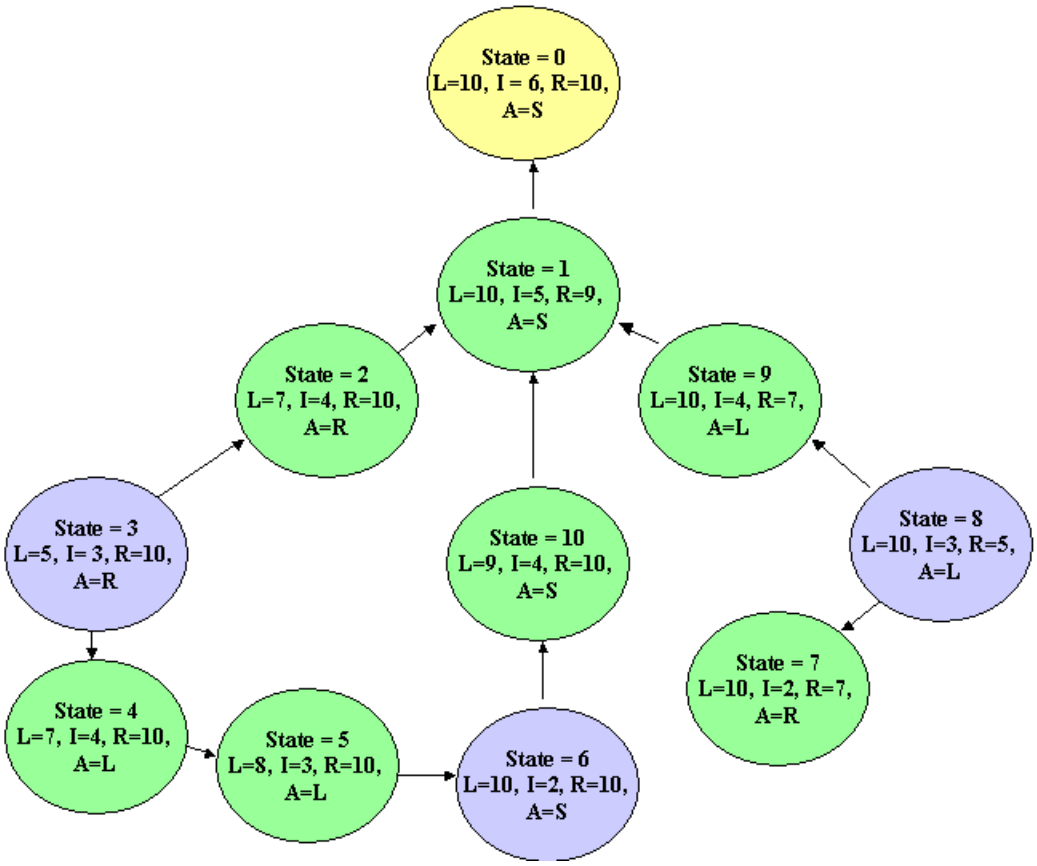

Figure 6.7. A directed graph of clusters from a recent memory. In each state there is a state number, which can be used as an index of an adjacency matrix, a left, L, and right, R, relative microphone parameters, and an intensity, I, parameter which indicates the intensity of the source. The yellow state is the goal state **G**. The purple states represent starting states of example paths.

The training set developed by the above process is written out in the same format as that used in the principle parts learning scheme discussed in the previous chapter. By doing this, the same GA and FFNN algorithms can be used.

The resulting FFNN developed using this method was tested on a simulated following robot in the simulator system developed for this research. Its following ability was as good as any of the others tested so far. Results are presented in the next section.

6.4 Experimental Results

FFNNs developed using the sensor prediction and graph learning algorithms were tested using primarily the same testing method discussed in section 5.7. The sensor prediction system created a reactive FFNN while the FFNN created using the graph learning was non-reactive. As a point of interest, working FFNN controllers were also created using the sensor prediction learning method with data collected onboard the mobile robots. All FFNNs created using graph learning used recent memories collected in the simulator

using the random but purposeful controller. Figures 6.8 and 6.9 show the results of the following test using the sensor prediction method.

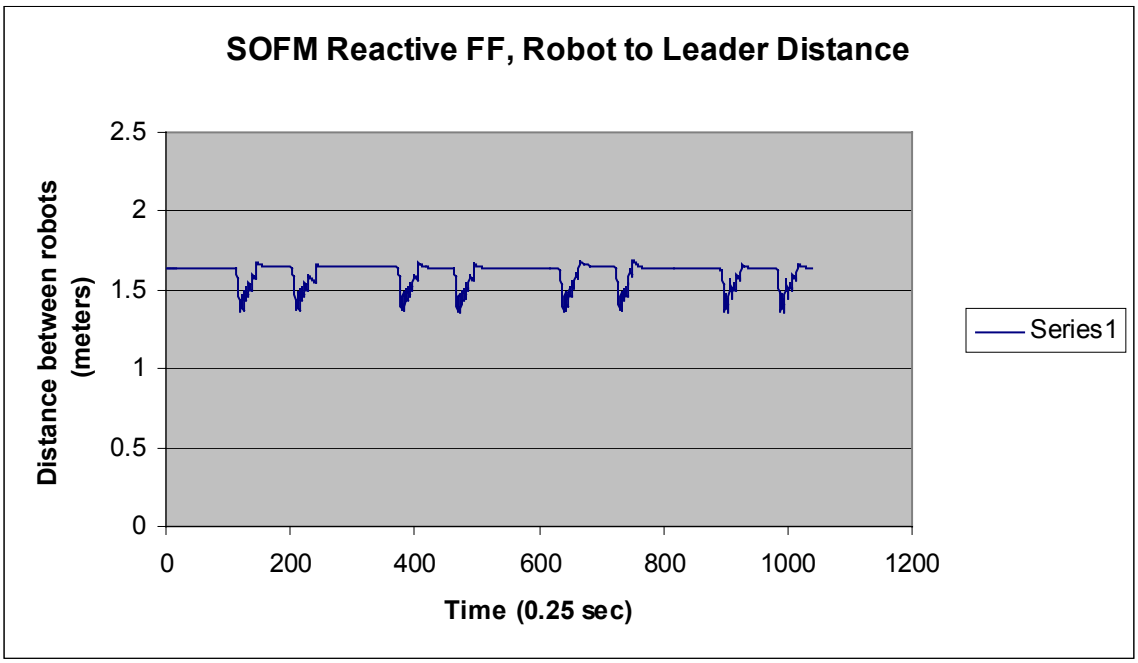

Figure 6.8. The robot to leader distance during a two lap following test in the simulator. The follower robot was controlled by a reactive FFNN created using the sensor prediction learning system.

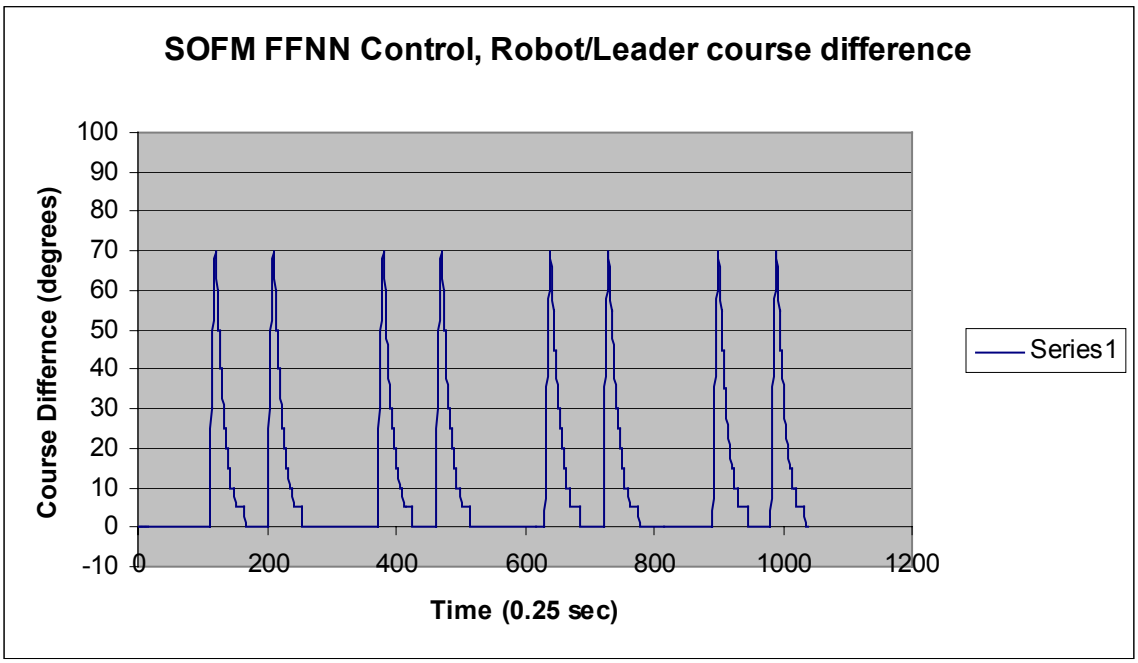

Figure 6.8. The robot to leader course difference during a two lap following test in the simulator. As in figure 6.8 the follower robot was controlled by a reactive FFNN created using the sensor prediction learning system.

As in the previous chapter the fluctuations in the distance plot occur because the leader is making a turn. They correspond in time with the peaks in the course difference plot. These results show that the robot was able to hold a stable distance from its leader, and the course was free of oscillations.

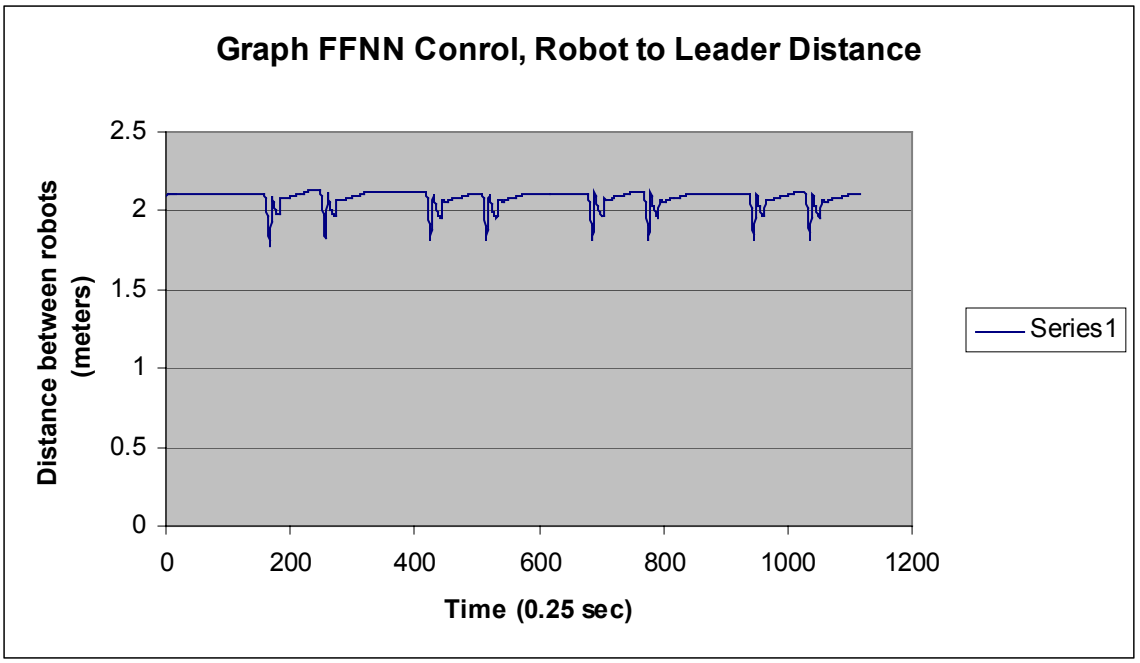

Figure 6.10. The leader/follower distance for a follower robot using a non-reactive FFNN created using the graph learning algorithm.

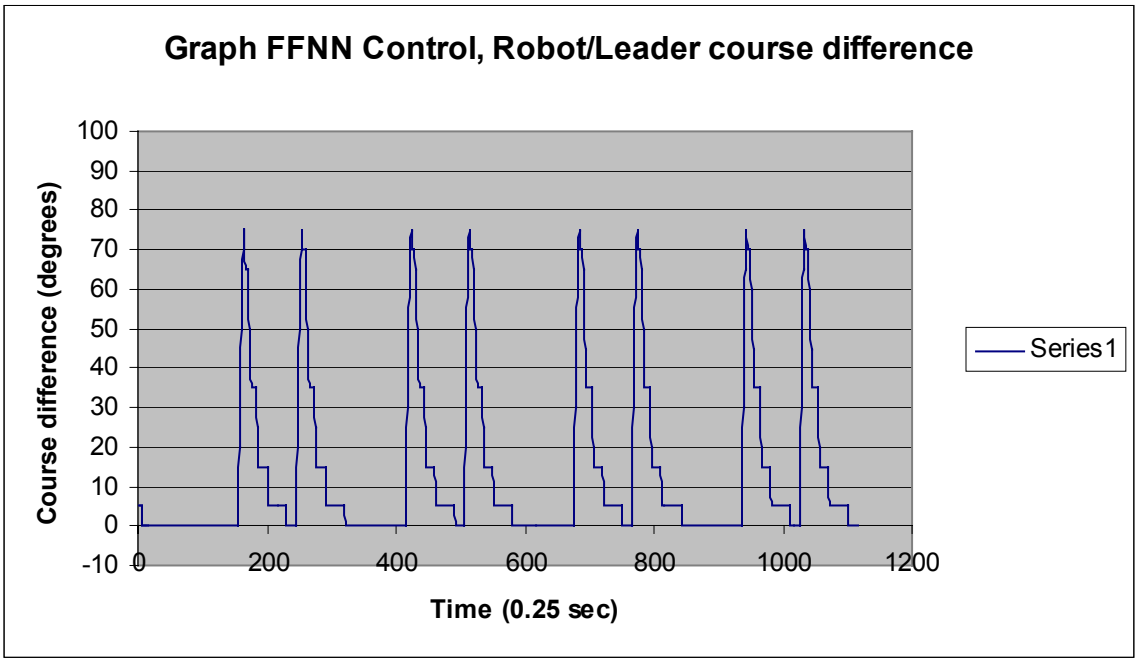

Figure 6.11. The leader/follower course difference for a follower robot using a non-reactive FFNN created using the graph learning algorithm.

Comparing the sensor prediction controllers test results to the reactive methods shown in chapter 5, it can be said that the sensor prediction method is at the least their equal. It is smoother in operation than the classic logic method, and offers a quicker reaction time as can be seen by comparing the maximum course differences of the two methods; the maximum course difference for the sensor prediction learning method never exceeded 70 degrees, while the reactive FFNN exceeded 70 degrees in half of the turns in the two lap test.

As a learning method, the graph learning algorithm is a step up from the principle parts method because it does not rely on rules of thumb applied at each time step to create training examples, thus lessening the reliance on a-priori knowledge. It is better than the sensor prediction based method discussed in the beginning of this chapter because it does not estimate results to rank solutions, it uses what actually worked and learns from that. While somewhat similar in function to Q Learning, a key merit of the graph method is that it creates a graph structure that can provide functionality in addition to learning. This functionality, briefly mentioned in the first chapter, is the subject of the Future work section in the final chapter.

Figures 6.10 and 6.11 above were made using a non-reactive controller generated using the graph learning algorithm. Again the curves show that the operation of the robot is stable and without oscillations. Compared to the principle parts results or behavior methods from chapter 5, the graph controller appears to be much more stable. This can be seen by the smoother shaped down slope on the peaks in figure 6.11 as compared to those of 5.18. The following distance for this algorithm is a little larger than the others. It is believed that this is due to the gain settings on the sensors used for this test.

Rucci, Wray, and Edelman[34] built a robotic barn owl in order to study the neurbiological structures responsible for localization behavior in owls of auditory and visual targets. Their goal was twofold, first they wanted to gain a greater understanding of how the various nervous system structures of a real owl worked, and second, they wanted to provide an alternative to current calibration techniques of equipment by developing a system that could operate in different sensor modalities. The neural network in the robotic owl was trained using a reinforcement learning algorithm (Q Learning is a type of reinforcement algorithm) with the weights being updated using a modified version of Hebbian learning[35]. They report good results after training for a few hours during which about 15,000 targets are presented to the system. Their system showed adaptability in the face of changing environments, but unlike the algorithms described in this chapter and those in chapter 5, there is no emphasis on reduction of physical motions of the robot.

Tesauro[61] created a program called TD-Gammon that plays backgammon at the world class level. It uses Q Learning to modulate error assignment parameters in order to train a multiple perceptron neural network. It learns to play by playing games against itself. Initially the moves are completely random, so the first games take several times longer than the usual 50 to 60 moves required by human players. As it learns, the length of the games is reduced to human like levels. In the latest version, it played 1.5 million games

in order to reach the world champion level. The program has been entered in backgammon tournaments and has been competitive at the world champion level. While it uses a reinforcement learning strategy to train neural networks as do the algorithms presented in this chapter and the previous one, its input space is constricted to a discrete non changing environment. And as with the work done by Rucci, Wray, and Edelman[34], there is no effort to reduce repetition during the learning process.

6.5 Summary

This chapter described two unsupervised methods that relax the cause and effect assumption used by the reactive learning and principle parts methods detailed in chapter 5. Two different clustering techniques were used to separate the recent memory into a collection of experiences or situations. The first learning method used an unsupervised learning algorithm to train FFNNs to select correct actions based on predicted sensor readings while the second method relied on a directed graph to preserve the topological information in the recent memory. Test results for both the sensor prediction and graph learning methods were shown. The cluster graph method holds promise to provide more functionality given further study. The next chapter summarizes the research done for this dissertation and suggests some methods to consider for future work.

Chapter 7. Summary and Future Work

From the outset, the goal of this research was to develop a methodology for controlling robots that would enable them to function without an abundance of a-priori knowledge embedded in either databases, maps, models, or rule systems. Furthermore, since the target application was a physical system, the method developed needed to avoid undue physical trials of solutions, so that time could be saved and mechanical systems not be overstressed. Although the goal of this research was to produce a general methodology, there was a specific test application in mind, one that was of interest to both the NRL and the robotics community, autonomous vehicle/robot formation maneuvering.

The work in this dissertation began with a study whose purpose was to make sure that the envisioned research would benefit the target application. The plan was to show that formation maneuvering using acoustic relative navigation was possible in simulation and that FFNNs were viable controllers in this task. This was important because NRL has an interest in autonomous underwater water vehicles and the properties of the undersea environment do not favor central control, GPS navigation, etc. or high bandwidth communications such as radio or wireless networks[7]. FFNNs were selected because they fit well with the goal of having an automatic system, i.e. they are more easily produced by other programs than alternate approaches that use symbolic methods. By selecting FFNNs, the complexity of ensuring that functions generated by symbolic methods are syntactically correct and are safe to run on the computer can be avoided. In the initial study, a method for training the FFNN using a GA that found both weights and transfer functions was developed and greatly benefited the development by eliminating the need for classifying the nature of the solution by hand picking the neural network's transfer function. A review showed that this technique is not discussed in common AI and machine learning texts or commonly described in the literature, however passing references to similar techniques were found in the literature[40 41].

Once it was shown that simulated robots were capable of formation maneuvering using feed forward neural networks the next logical step was to transition the work to the physical world. Because UUVs were not available, mobile robots fitted with an ad-hoc acoustic communication system were used. Various problems associated with the formation-maneuvering task were tackled in an incremental fashion. To facilitate navigation using leader/follower paradigm, a low bandwidth frequency multiplexing communication scheme was developed in which each follower robot navigated based on the intensity of chirps transmitted by its assigned leader. The lab computers controlled the lead robot while the others used acoustic sensors and relative navigation to maintain the formation. Three methods of control were used, a baseline system, a behavior based system and one based on FFNNs. The end result was that the lab's three mobile robots successfully maneuvered around the lab in a line formation, showing that relative navigation using acoustic communications was a viable approach for formation maneuvering tasks. In the literature, FFNNs are not specifically used for formation maneuvering tasks, and since most formation maneuvering work is targeted at mobile

robots, not UUVs, they commonly use other types of navigation systems such as vision, infrared, or wireless, not relative acoustic navigation.

Having shown that the acoustic relative navigation systems and FFNNs were adequate for the task, the focus of the work shifted towards the general goal of developing a method for controlling robots in unstructured environments without requiring the robot to physically execute all trial solutions. In order to do this, methods used to train the FFNNs were automated. The task was separated into three subtasks:

- Collection of a recent memory in which there is enough information to create a "draft", or first cut version of a controller.
- Extraction and formatting of the information in the recent memory so that it can be used to train a FFNN.
- Train the FFNN using the GA process.

For the robot to avoid extensive physical execution of possible solutions, it learned from a recent memory in which another controller was operating. The other controller could be a baseline controller such as the Braitenberg controller discussed in chapter 4, or a random controller. The idea was to collect data so that a FFNN could learn to control the robot from data that included various environmental influences. The process was envisioned as an iterative process in which the controllers could be improved as time passed, but not as one in which the robot physically executed each and every candidate solution.

Key to the leaning process is collection of data that can be learned from. In order to facilitate this task, the random but purposeful controller concept was created. It tied goals to data collection in order to collect an adequate group of examples from which algorithms could learn, but it did not attempt to associate sensor patterns and decisions with the goals. In order to make sure that the most could be made from a data set, simple symmetry rules were applied, i.e. sensor mirroring, in order to ensure a more complete example set. These straightforward concepts greatly improved the progress of this work, and are not commonly or at all discussed in texts or literature.

Once an adequate recent memory was available, two supervised learning techniques were developed. The techniques differed in that one was reactive, it only considered current sensor readings, and the other was non-reactive, it considered a finite number of past values. Because only current values are considered in the reactive scheme, sensor/action pairs (positive learning examples) can be taken out of context based on if they helped further the predetermined goal. Using this strategy training example files were created and the GA was used to create a FFNN which controlled both simulated robots and the lab robots in formation maneuvering tasks.

Since the non-reactive system needs to consider contiguous groups of sensor/action pairs, the earlier strategy would not suffice. The principle parts routine was created to overcome this restriction. In this routine the sensor defined goal parameter was viewed as a time series that could be broken down into a group of principle parts. These parts

were: trends in which the goal parameter was rising, dropping, remaining stable, or fluctuating. Extracting sequences of sensor/action pairs in which the action remained constant and the goal parameter either rose or remained stable formed training examples. Examples from the dropping set were used only if they overlapped or were adjacent to an example from the rising example set. The action used for the falling set was that of the adjacent rising example. As in the reactive case, an example file was created and a FFNN was trained. In simulation, the FFNN trained using this procedure was able to distinguish between a sound source coming directly towards it and one moving directly away from it, something that cannot be done with a reactive method.

The problem with the above approaches is that they require a supervisor system that analyzes the recent memory at each time step. While this works, and worked well in the target application, in order to provide a more general solution this restriction was dropped in favor of an unsupervised approach. Two different algorithms that relied on clustering the data into libraries of experiences were developed. The first technique used a Kohonen SOFM to do the clustering and relied on sensor prediction to rank solutions. This technique also used sensor mirroring at both the data level and cluster level. Using this technique reactive FFNN were developed from both simulated data and data collected using the lab robots.

The second unsupervised technique, the graph learning method described in the last section of the previous chapter, is based on the application of graph theory and is the culmination of the work in this dissertation. It fulfills the original goals set out for this dissertation in that it allows a robot to function without a-priori knowledge embedded in either databases, maps, models, or rule systems. All that is required is a description of the goal state in terms of sensor/action values. It allows the system to avoid undue physical trials of solutions, by learning from previously experienced situations, and using techniques such as sensor mirroring. It shares some similarities with Q Learning, but is different in that the table built in Q-Learning does not preserve information on the sequence of sensor/action relationships to one another. The graph-based technique organizes the situations experienced in the recent memory into a data structure that can work in conjunction with a FFNN. Besides providing an effective means to learn, the graph will allow features not usually associated with Q-Learning including tracking of progress towards goals, prediction of sensor values based on actions, selection of alternate routes, and prediction of non-experienced states. For this dissertation this learning method was tested in simulation with good results, as seen in chapter 6.

Table 7.1 below summarizes the progression of algorithms developed in this work. Starting with the reactive algorithm the need for human-in-the-loop was eliminated, while each proceeding algorithm adds capability or removes the need for the application of a-priori knowledge.

Table 7.1 below summarizes the attributes of the learning algorithms developed for this research.

Table 7.1 This table shows the attributes of the learning algorithms developed for this dissertation. The attributes of Classic Logic and Behavior methods are not shown here because they do not learn. The Onboard Robot Estimation technique is the method described in chapter 4 in which the robot was programmed to estimate the direction of the sound source based on what training data was provided by the opeator/techer.

Attribute	Multi-robot Simulator	Onboard Robot Estimation	Reactive	Principle Parts	Sensor Prediction	Graph Learning
Type of Learning	Unsupervised	Supervised	Supervised	Supervised	Unsupervised	Unsupervised
Human in the loop	No	Yes	No	No	No	No
Physical repetition	Yes	No	No	No	No	No
Test Environment	Simulation	Simulation, Mobile Robots	Simulation, Mobile Robots	Simulation	Simulation, Used Robot Data	Simulation
Learning Data collection method	Physical Trial	Teacher	Operator driven loop, Classic logic	Purposeful but Random Controller	Operator driven loop, Classic logic	Purposeful but Random Controller

In summary the specific contributions of this research include:

- A GA method of training FFNNs that incorporates finding the transfer function types for the hidden and output layers was developed. This is important because it is not always obvious to the researcher which transfer functions are needed for a given problem.
- Formation maneuvering using FFNN control was shown to viable in both simulation and using mobile robots. Most formation maneuvering controllers rely on control laws, behavior schemes, potential fields, or virtual structure approaches. This is important because FFNNs offer a flexible, machine generatable method of control.
- Formation maneuvering using acoustic communication systems was shown to be effective in both simulation and on mobile robots. Most robots that operate in formation use radio, video, or infrared communication systems. While these work well, they are not well suited for the target application in mind, Unmanned Underwater Vehicles.
- A method of exploring the environment that is tied to goals, dubbed the 'purposeful but random controller' was developed. This is important because

random explorations, such as those used in Q Learning, can take many time steps to piece together enough data to form a usable operating strategy.

- A method for extending the data collected in exploration, called 'sensor mirroring', was developed based on symmetry of sensors. Using this technique, what occurs on one side of the robot does not have to occur on the other side of the robot for it to be learned.
- A training method based on the clustering of recent memories and sensor prediction that provided excellent results in the target application was developed. This method is an improvement over other methods because it does not require the robot to execute trial solutions and it is unsupervised so no assumptions or rules of thumb are required to provide training examples.
- A graph based method of learning that shares some features with Q Learning was developed. Besides performing well on the target application, the main strengths of this algorithm are that it does not require the robot to execute trial solutions, it is unsupervised, and its graph structure can be used to provide tracking of progress towards goals, prediction of sensor values based on actions, selection of alternate routes, and prediction of non-experienced states.

The last bullet refers to several other features that the graph based method of learning can provide. These features are the primary topic discussed in the next section, Future Work.

7.1 Future Work

This dissertation had two distinct parts, an applied part in which significant time and effort were spent getting the first algorithms to work on mobile robots, and a learning/theoretical part. It is readily acknowledged that both areas present many opportunities for further study and improvements. For example, the acoustic systems used in the formation maneuvering work can be improved by finding better frequencies, improved chirping, use of baffles to improve directivity, etc. The algorithms discussed in chapter 6 need to be integrated into a seamless architecture on board the robots so that no human intervention is required and the system needs to be tested in the underwater environment. Scaling and clustering techniques used in the learning algorithms need to be studied and improved. But, from a conceptual viewpoint, the most inviting opportunities lie in the graph structure used in the graph-based learning algorithm.

The following bullets describe how the topological relationship provided by this structure could be used to provide capabilities that work in conjunction with the learning algorithm.

- The graph can be used to trace the progress of an agent as it moves about in its environment. The agent's current state (situation and state are analogous here) can be determined by comparing current sensor value to those that comprise the nodes of the graph. Once it is known where in the graph the agent lies, the results of its actions can be observed. The graph becomes an observer routine's key tool in determining if the agent's actions are having the desired results.
- By defining which situations are desirable, the sequence of actions from all situations to desirable situations can be found. When the observer notices that a

controller is caught in a local minima (it is in a state in which its sensor information is leading it to repeatedly make incorrect decisions) it can override the controller and select actions that lead to more desirable states.

- Once it is known where in the graph the agent lies, the results of its actions can be predicted using the graphs connectivity. These predictions can be used for other functions such as planning of future actions and maneuvers.
- By analyzing the composition of the nodes, and using properties such as symmetry, interpolation and extrapolation, the existence of non-experienced nodes can be postulated. These nodes can be added into the graph and their legitimacy can be confirmed with a more focused exploration of the solution space. This functionality provides a rudimentary form of reasoning that is more focused than earlier brute force methods described such as sensor mirroring.

The next section provides a brief conceptual overview of each of these features. It begins with an explanation of how the graph can be used as part of an observers memory, continues with discussions on avoiding local minima sensor prediction. Finally some concepts of how the graph structure can be used to perform rudimentary reasoning are described.

7.1.1 Next State Prediction and Observer Memory

The situational memory provided by the directed graph developed for the graph learning method is an untapped resource that holds the promise of providing extra functionality. In order to make use of the graph there would need to exist an onboard architecture in which the aforementioned features could be integrated.

In the previous chapter a directed graph was used to help train a FFNN that enabled the robot to navigate in its environment. But in order to monitor its progress the robot needs a feedback process in which actions are compared with desired results. The graph used to train the FFNN provides a good starting point because its structure describes what usually occurs given a certain sensor state and action taken. By tracing the robot's progress through this structure an observer routine can determine if the robots actions are having the required results.

In the learning algorithm the nodes of the graph were comprised of sensor/action pairs and edges existed between the nodes if they were adjacent to each other in time. In order to illustrate the use of the graph in a state monitoring/prediction task, the composition of the graph is changed so that the nodes reflect only sensor states and the edges indicate an action that was required to get from one sensor state to another. The graph, now a state machine, could be used to predict sensor states based on actions. Figure 7.1 shows a graph whose nodes are sensor states and the edges are actions. Note that the graph is not complete, but by looking at state 3 and following the edges labeled S (Straight) the progression of sensor values can be traced.

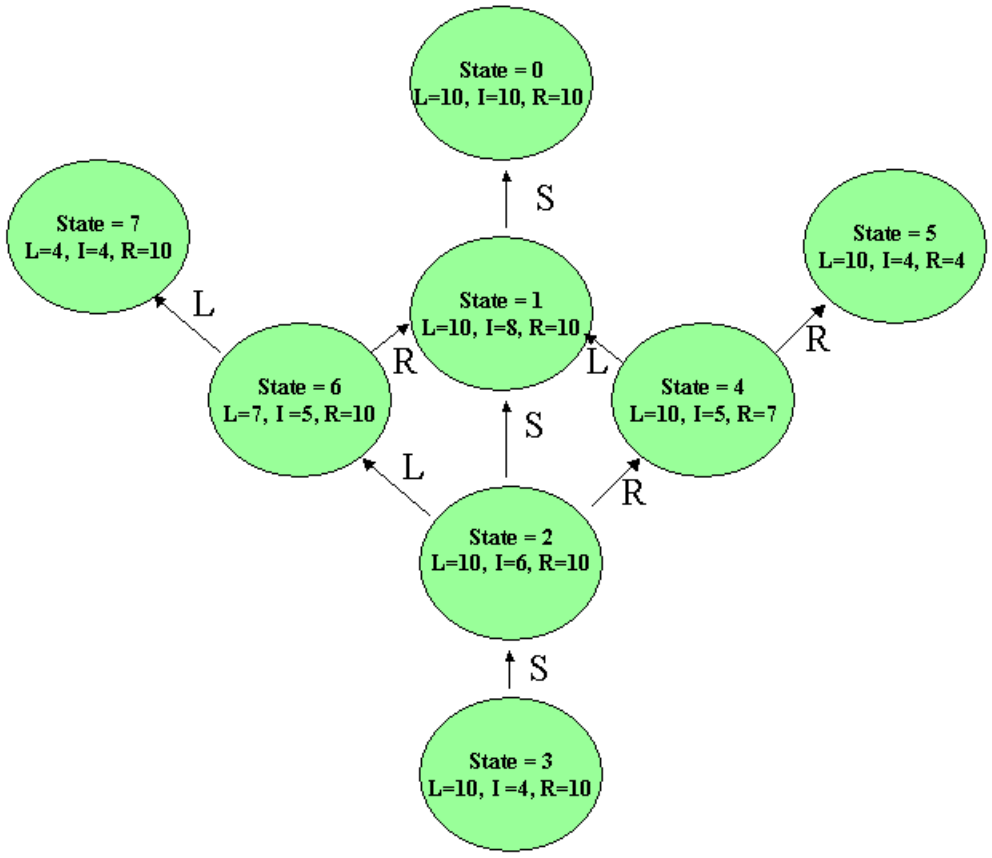

Figure 7.1. A graph for use in predicting upcoming sensor readings. Each state has left (L) and right (R) relative sensor values and a sensor intensity (I) value. The edges indicate the actions (S= straight, L = left, R = right) that cause transitions from state to state.

Consider the following example. If the observer determines from the robot's current sensor readings that it is in state 2, and the FFNN directs the robot to go straight, the observer should see the robots sensors change from their current state to that of the node labeled state 1. If this does not happen, then at the very least the observer can note the unexpected event. Increasing numbers of unexpected events may indicate any number of conditions, such as degrading sensor performance, change in environment, or the FFNN currently being used is out of date. By using the graph to monitor progress the observer routine can take action to warn the users, or correct the situation.

7.1.2 Observer Control and Alternate Paths

There may be states in which the FFNN does not work correctly, because the FFNN represents the global solution for the states for which it was trained. But recall that it is trained with a filter created from the set of states that come from the shortest paths to the stable state. States that were not in the filter will not have been used to train the FFNN, but they still exist in the environment, and are represented in the graph. When these states are encountered the FFNN may or may not guide the robot towards the optimal state. In the event that it does not, it may be possible to use the graph to find a series of

actions that can get the robot back into a state in which the FFNN can make the correct decisions.

For example, as was earlier described, one of the main reasons that FFNNs that consider the past were studied was so that the robot could tell the difference between heading directly towards and directly away from the sound source. While this was tested with encouraging results, it can be said that since the FFNN is a global solution to the training examples, there is no guarantee that it will react correctly to this situation. But if the situation is present in the training set, and therefore present in the graph, the graph could be used to guide the robot out of this situation. In the illustration in figure 7.2 below, the robot is in state 3, the graph node highlighted in blue. At this point an observer routine could have noted slow loss in energy from previous states. It could be surmised that the FFNN is not making the correct decisions in this situation. The edges that are highlighted in red indicate an alternate path in which energy is being gained instead of lost. The observer could take over control of the robot from the FFNN and use the graph to guide the robot until the FFNN can take over again.

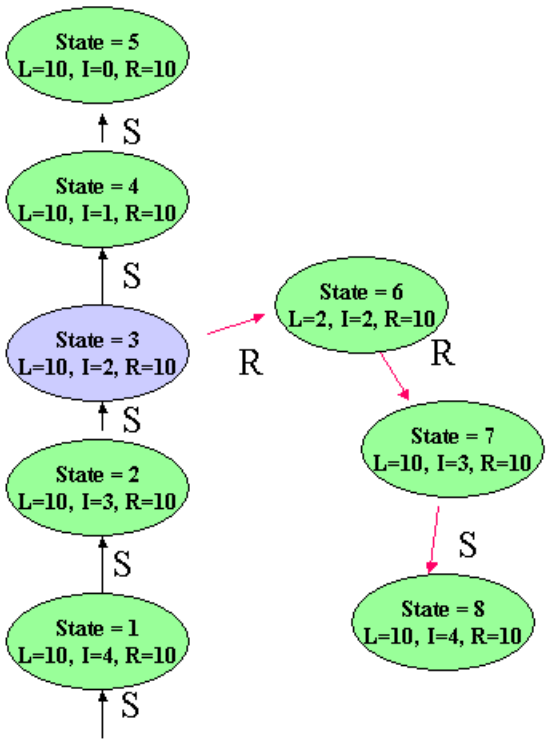

Figure 7.2. In this figure the robot is in state 3. From state 1 to state 3 there has been a steady loss of energy. Noticing this loss, the observer could take over command of the robot and use the actions on the edges highlighted in red until the FFNN is ready to resume control.

7.1.3 Using Unseen States for Rudimentary Reasoning

One of the major problems with all the learning routines discussed so far has been that almost without exception, in order to learn something, it had to be experienced. The learning routines developed here allow the learning from recent memories so that sequences of actions do not have to repeated until the desired result is obtained, but somewhere in the memory, the desired result must be present. But humans and animals can learn without having to directly experience each thing that is learned. For example, when children are learning to pound nails into boards they learn quickly that it hurts, and hurts a lot, when they hit their thumb with the hammer. Usually they hold the nail with their thumb and the index finger and they know if they were to hit their index finger with the hammer it would also hurt. They know that if they had switched hands with the hammer and the nail and hit the thumb on the other hand, it too would hurt. So it stands to reason that certain assumptions are being made.

Symmetry was being used in the above example when the children decided that it would hurt if they hit their thumb on the opposite hand. Other primitive assumptions may also be useful, such as interpolation and extrapolation. For example assuming that the index finger would hurt may be an extrapolation based on hitting the thumb, or if the thumb had been hit and the index finger had been hit, knowing the index finger would also hurt could be thought of as an interpolation.

Sensor mirroring, described in chapter 5, was a broad-brush approach to this problem that made assumptions based on symmetry. It helped to reduce the size of the memory that had to be experienced in order for a following robot to learn to turn both directions at the right time. Using the graph, and the primitive rules of symmetry, interpolation, and extrapolation, the time required to explore the environment could be decreased. Figure 7.3 below shows a graph that has had nodes inserted into it that were not experienced. These nodes could be marked so that when they were experienced, the observer could confirm that they were correct.

Looking at the graph in figure 7.3, 7 of the 13 total nodes were in recent memory. Assuming that the rules of symmetry, interpolation, and extrapolation were applied conservatively, the robot could have made an almost 50% reduction in environmental exploration. This tactic, postulating the existence of a state based on previously experienced states and basic rules, forms a rudimentary method of reasoning that may be able to help machines save time, and wear and tear in the future.

In closing, the goal of this research was to develop a technique/system that could produce situation dependent control algorithms for a robot in near real time without requiring an abundance of a-priori knowledge in data bases, maps, models etc, or by requiring them to make multitudes of physical trials. The progression of algorithms developed, culminating with the graph based leaning technique, have been shown to fulfill these goals and provide future capabilities including observers, situation/action based prediction and rudimentary reasoning.

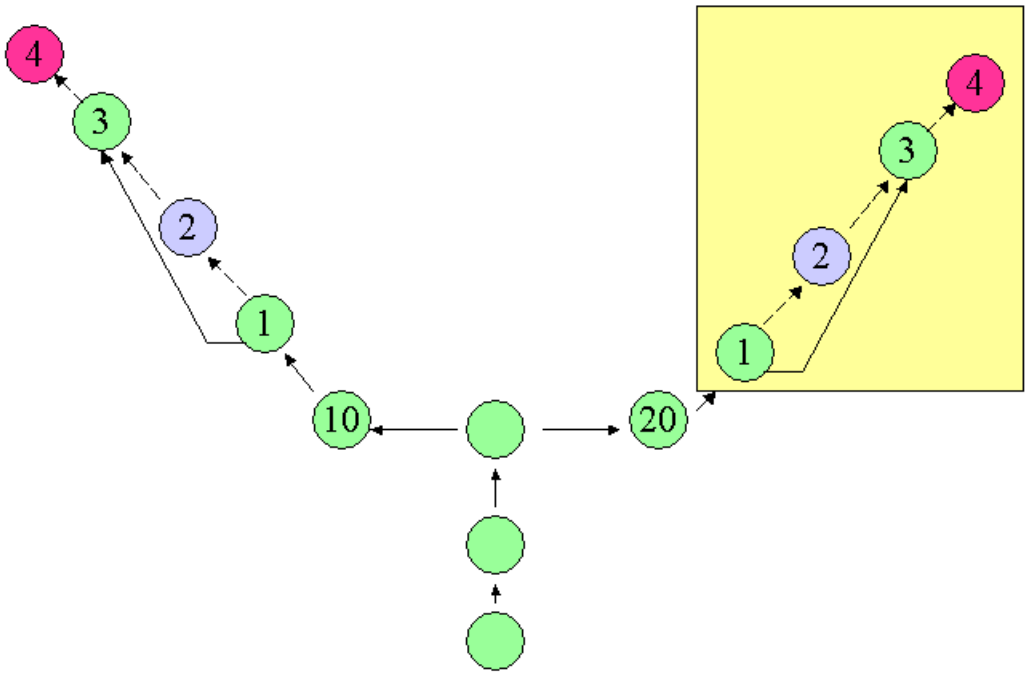

Figure 7.3. A situation graph that was built from a recent memory. The green nodes, excluding those in the yellow box were present in the recent memory. The arrow from state 1 to 3 is what was experienced. Using interpolation, node 2 was inserted, and using extrapolation, node 4 was inserted. Using symmetry, a symmetric copy of nodes 1 to 4 was built and connected at node 20. The yellow box around those nodes denotes that they are nodes created using symmetry.

References

1. McFarland and Bosser, *Intelligent Behavior in Animals and Robots*. Cambridge Ma.: MIT Press, 1993, page 6.

2. McFarland and Bosser, *Intelligent Behavior in Animals and Robots*. Cambridge Ma.: MIT Press, 1993, pp 257-279.

3. Nancy Forbes, "Biologically Inspired Computing," http://www.egr.msu.edu/~hujianju/bio-inspired_computing.htm

4. Grefenstette, J. J. and Schultz, A. C. , "An evolutionary approach to learning in robots," *Machine Learning Workshop on Robot Learning,* New Brunswick, NJ, 1994

5. Tom Mitchell, *Machine Learning*. McGraw-Hill, 1997, pp 367-386.

6. Brooks, R. A., "Challenges for Complete Creature Architectures," *First International Conference on Simulation of Adaptive Behavior*, Paris, France, September 1990, pp. 434–443.

7. Brian Bourgeois, Patrick McDowell, "UUV Teams for Deep Water Operations," *Underwater Intervention 2001*, New Orleans, La., February 2002

8. J. Desai, J.P. Ostrowski, V. Kumar, "Controlling Formations of Multiple Mobile Robots," *Proceedings of 1998 IEEE Int. Conf. on Robotics and Automation*, 1998, pp. 2864-2869

9. McDowell, Bourgeois, Iyengar, "Formation Maneuvering Using Passive Acoustic Communications," *2004 IEEE Int. Conf. on Robotics and Automation,* New Orleans, LA, April, 2004

10. Ronald C. Arkin, *Behavior Based Robotics*, MIT Press, 1998, pp. 10-14

11. Aho, Hopcraft, Ullman, *The Design and Analysis of Computer Algorithms*, Addison Wesley 1974, pp. 67–69

12. M.A. Lewis, K-H. Tan (1997). "High Precision Formation Control of Mobile Robots Using Virtual Structures," *Autonomous Robots*, volume 4, Springer 1997, pp. 387-403

13. J. Desai, V. Kumar, J.P. Ostrowski, "Control of Changes in Formation For a Team of Mobile Robots," *Proceedings of 1999 IEEE Int. Conf. on Robotics and Automation*, 1999, pp. 1556-1561

14. J. Desai, J.P. Ostrowski, V. Kumar, "Modeling and Control of Formations of Nonholonomic Mobile Robots," *IEEE Trans. on Robotics and Automation*, vol. 17, no. 6, December 2001

15. R. Fierro, A. Das, V. Kumar, and J. P. Ostrowski, "Hybrid Control of Formations of Robots," *IEEE Int. Conf. on Robotics and Automation*, Seoul, Korea, May 2001, pp. 157-162

16. T. Balch, R. Arkin , "Behavior-Based Formation Control for Multirobot Teams," *IEEE Trans. on Robotics and Automation*, vol. 14, no. 6, December 1998

17. N.E. Leonard, E. Fiorelli, "Virtual Leaders, Artificial Potentials and Coordinated Control of Groups," Proceedings of the 40th IEEE Conf. on Decision and Control, December 2001, pp. 2968-2973

18. T.R. Smith, H. Hanbmann, N.E. Leonard, "Orientation Control of Multiple Underwater Vehicles With Symmetry-Breaking Potentials," *Proceedings of the 40th IEEE Conf. on Decision and Control*, December. 2001, pp. 4598-4603

19. H. Yamaguchi, T. Arai, "Distributed and Autonomous Control Method for Generating Shape of Multiple Robot Group," *Proceedings of IEEE Int. Conf. on Intelligent Robots and Systems*, 1994, pp. 800-807

20. Brian Bourgeois, Patrick McDowell, "UUV Teams for Deep Water Operations," *Underwater Intervention 2001*, New Orleans, La., February 2002

21. University of Reading, "Flocking Robots", http://www.cyber.rdg.ac.uk/CIRG/robots/flocking.htm

22. Ellen Thro, *Artificial Life Explorer's Kit*, Sam's Publishing 1993, pp. 105-106

23. Erann Gat, "On Three Layer Architectures," *Artificial Intelligence and Mobile Robots*, David Kortenkanp, R. Peter Bonnasso, Robin Murphy, eds., AAAI Press

24. Brooks, Rodney A. "A Robust Layered Control System for a Mobile Robot", *MIT AI Lab Memo 864*, September 1985

25. Iyengar, Graham, Hegde, Graham, "A concurrent control architecture for Autonomous Mobile Robots using Asynchronous Production Systems", *Automation in Construction*, vol 1, 1993, pp 371 – 401

26. Jirenhed, Hesslow, Ziemke, "Exploring Internal Simulation of Perception in Mobile Robots", *Eurobot 2001*, Lund 2001

27. Maes, P. and R. A. Brooks, "Learning to Coordinate Behaviors", *AAAI*, Boston, MA, August 1990, pp. 796—802

28. Gary B. Parker, "Punctuated Anytime Learning for Hexapod Gait Generation", *Proceedings of the 2002 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, EPFL, Lausanne, Switzerland, October 2002

29. Patrick McDowell, "Biologically Inspired Learning System," Proposal for Thesis Research, April 2003

30. Sutton and Bartow, *Reinforcement Learning: An Introduction*, MIT Press 1998, page 1.1

31. Vinay Shah, "Practical Implementation of the Sound Localization Algorithm on a Robot," Rensselaer Polytechnic Institute, Research Experience for Undergraduates, August, 2003

32. Aleksandr Kushleyev, Vaibhav Vohra, "Sound Localization," University of Maryland, 2004

33. Reeve, R. and Webb, B., "New neural circuits for robot phonotaxis," *Philosophical Transcations of the Royal Society*, 2003, pp 2245-2266

34. M. Rucci, J. Wray, G.M. Edelman, "Robust localization of auditory and visual targets in a robotic barn owl," *Robots and Autonomous System*, vol 30, 2000, pp 181-193

35. Hagan, Bemuth, Beale, *Neural Network Design*, Campus Publishing Service, University of Colorado, page 7.2

36. Tom Mitchell, *Machine Learning*, McGraw-Hill, 1997, page 105

37. Robert J. Urick, *Principles of Underwater Sound, 3rd Edition*, McGraw-Hill, 1983, page 100

38. Simon Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan, 1994, pp 18-26

39. Hagan, Demuth, Beale, *Neural Network Design*, Campus Publishing Service, University of Colorado, 1996, page 2-11

40. Nan Jiang, Zhiye Zhao, Liqun Ren, "Design of Structural Modular Neural Networks Genetic Algorithm," *Advances in Engineering Software*, vol 34, pp 17-24, January 2003

41. Xiaohui Hu, "PSO Tutorial", http://www.swarmintelligence.org/tutorials.php

42. Tom Mitchell, *Machine Learning*, McGraw-Hill, 1997, page 86

43. Brooks and Iyengar, *Multi-Sensor Fusion*, Prentice Hall, 1998, pp 185-187

44. Tom Mitchell, *Machine Learning*, McGraw-Hill, 1997, page 256

45. Barbara Webb, "What does robotics offer animal behavior?" *Animal Behavior*, volume 60, 2000, pp 545–558

46. Marvin Roe, Brian Bourgeois, Patrick McDowell, "Simulator Development for Multiple Unmanned Underwater Vehicles," *Proceedings of ACM Southeast 2003 Conference*, Savannah, GA, March 2003

47. Ingebret Gausland, "Physics of Sound in Water", *Proceedings of Seismic and Marine Mammals Workshop*, London 1998

48. Marshall Bradley, *Environmental Acoustics Handbook 2nd Edition*, 1996, page 23

49. Finn B. Jennsen, William A. Kupperman, Micheal B. Porter, Henrik Schmidt, *Computational Ocean Acoustics*, AIP (American Institute of Physics) Press, 1994

50. P. McDowell, J. Chen, B. Bourgeois, "UUV Teams, Control From A Biological Perspective", *Proceedings of the Oceans 2002 MTS/IEEE Conference*, Biloxi MS, pp 331-337

51. Ah-Hwee Tan, Samin Karim, Liz Sonenberg, "Reactive Plan Execution versus Reactive Learning: A Case Study on Minefield Navigaiton", *IAT*, France, 2005

52. Aho, Hopcraft, Ullman, *The Design and Analysis of Computer Algorithms*, Addison Wesley 1974, pp 50–52

53. Laurene Fausett, *Fundamentals of Nerual Networks Architectures, Algorithms, and Applications*, Prentice Hall, 1994, page 425

54. Aho, Hopcraft, Ullman, *The Design and Analysis of Computer Algorithms*, Addison Wesley, 1974, pp 50–52

55. Simon Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan, 1994, pp 408–422

56. "Kohonen Self Organinzing Feature Map Demo", http://www.etimage.com/java/ai/somf/sofm.htm

57. Dehghan, Faez, Ahmadi, Shridhar, "Handwritten Farsi (Arabic) work recognition: a holistic approach using discrete HMM", *Pattern Recognition*, vol 34, 2001

58. Aaron Tanenbaum, Yedidyah Langsam, Moshe Augenstien, *Data Structures using C*, Prentice Hall, 1990, page 513

59. Tom Mitchell, *Machine Learning*, McGraw-Hill, 1997, page 239

60. Aho, Hopcraft, Ullman, *The Design and Analysis of Computer Algorithms*, Addison Wesley, 1974, pp 67–69

61. Gerald Tesauro, "Temporal Difference Learning and TD-Gammon", *Communications of the ACM*, Vol. 38, No. 3, March 1995

Appendix: Feedforward Network Description File

The following text is the file that contains the weights and various parameters used to reconstruct a feed-forward network in the robot school.vi program.

Feed-forward network weights.
Network id
2
Transfer function (0-discrete, 1-linear, 2-discrete)
0
Number inputs
2
Hidden layer size
4
Number outputs
2
Weights from input layer to hidden layer node 0
 -0.804000 0.343000
Weights from input layer to hidden layer node 1
 -0.279000 0.497000
Weights from input layer to hidden layer node 2
 0.965000 -0.719000
Weights from input layer to hidden layer node 3
 0.511000 0.321000
Weights from hidden layer to output layer node 0
 0.640000 -0.010000 -0.908000 0.852000
Weights from hidden layer to output layer node 1
 -0.396000 -0.555000 0.440000 0.247000
Transfer functions.
 0 0 0 0
End of network, good-bye

Vita

Patrick McDowell was born in St. Louis, Missouri, on August 25, 1962. He received his bachelor's degree in computer science in 1984 from the University of Idaho. In 1995, while working as a computer scientist, he began his graduate studies. In 1999 he received his master's degree in computer science from the University of Southern Mississippi and he began his doctorate studies at Louisiana State University as a Board of Regents Fellowship recipient. His research interests include legged robotics, machine learning, and artificial intelligence.